NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

JANUS/ADA SOFTWARE IMPLEMENTATION OF A
STAR CLUSTER LOCAL AREA NETWORK OF
PERSONAL COMPUTERS

by

Thomas Victor Works

December 1986

Thesis Advisor: Uno R. Kodres

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 54	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) JANUS/ADA SOFTWARE IMPLEMENTATION OF A STAR CLUSTER LOCAL AREA NETWORK OF PERSONAL COMPUTERS (U)

12 PERSONAL AUTHOR(S)
Works, Thomas V.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 December	157

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Local Area Network; Zenith 2400, Data Communications; CPM-86; Shared Resources; Process Coordination; JANUS/ADA

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis describes the detailed design and implementation of a star cluster local area network among multiple Zenith Z-100 microcomputers. The Z-100s are linked together by the Concentrator, a server system consisting of a power supply, an iSBC 86/12A single board computer, and three BLC 8538 eight channel I/O expansion boards, through RS232c USART ports. The local area network software consists of a server program resident in the Concentrator which provides the communication links between the Z-100s and the utility programs resident in each microcomputer. These utilities include file and message transfer (both point to point and broadcast), directory listing transfer, and online user identification. The program and utilities are written in JANUS/Ada, with assembly language sub-routines for machine specific functions, and are designed to run with the CP/M-86 operating system.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Uno R. Kodres	22b TELEPHONE (Include Area Code) (408) 646-2197 22c OFFICE SYMBOL 52Kr

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

JANUS/Ada Software Implementation of a Star Cluster Local Area Network of Personal Computers

by

Thomas Victor Works
Lieutenant, United States Navy
B.A., University of Rochester, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
December 1986

ABSTRACT

This thesis describes the detailed design and implementation of a star cluster local area network among multiple Zenith Z-100 microcomputers. The Z-100s are linked together by the Concentrator, a server system consisting of a power supply, an iSBC 86/12A single board computer, and three BLC 8538 eight channel I/O expansion boards, through RS232c USART ports.

The local area network software consists of a server program resident in the Concentrator which provides the communication links between the Z-100s and the utility programs resident in each microcomputer. These utilities include file and message transfer (both point to point and broadcast), directory listing transfer, and online user identification. The program and utilities are written in JANUS/Ada, with assembly language subroutines for machine specific functions, and are designed to run with the CP/M-86 operating system.

DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurence of a trademark, all registered trademarks appearing in this thesis will be listed below, following the firm holding the trademark.

Intel Corporation, Santa Clara, California:

INTEL MULTIBUS iSBC 86/12A

Digital Research Incorporated, Pacific Grove, California:

CP/M-86

National Semiconductor, Santa Clara, California:

PLC 8538

Zenith Data Systems Corporation, St. Joseph, Michigan:

Z-100

Xerox Corporation, Stamford, Connecticut:

ETHERNET

RR Software, Inc. Madison, Wisconsin:

JANUS/Ada

United States Government, Washington, D.C.:

ADA
ARPANET

Bell Laboratories, Murray Hill, New Jersey:

UNIX

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I wish to thank Dr. Uno Kodres, Mr. Mike Williams, and Mr. Russ Whalen, without who this thesis would not have been possible.

I wish to thank my family and friends for being there when I needed you.

I wish to thank Kathleen, for whom I dedicate this thesis and all my accomplishments, for it is her love, patience, and understanding that makes it all possible.

I. INTRODUCTION

A. BACKGROUND

A critical requirement for today's office and laboratory environments is the sharing of expensive data and resources. A network of microcomputer workstations linked together can share data by transmitting files and messages between users and share pheripheral resources such as printers by scheduling users in the most efficient manner. In the laboratory environment in particular, it is especially important and desirable to share data, experiments, and software development among lab members and to provide the means to distribute data to all members from a single source such as the professor or laboratory supervisor. Broadcast transmission from one workstation to all workstations of files or messages, or using one workstation as a 'bulletin board' to store messages over a period of time for access by all members, both serve this purpose. A testbed for research in data and resource sharing in the laboratory environment is a MULTIBUS based computer configuration which permits a single board computer to communicate with three input/output boards each containing eight RS-232 ports. This configuration serves as a concentrator for a star cluster local area network, which can connect up to twenty four RS-232 compatible devices.

This thesis is, in many ways, a companion to the thesis done by Lt. Cmdr. Robert Hartman and Capt. Alec Yasinsac,

[Ref. 1]. Their thesis involved the implementation of a prototype star cluster local area network of microcomputers connected to a Vax 11/780 computer over the ETHERNET communications device. The Vax 11/780 system operating under the UNIX operating system provides access to the ARPANET wide area network. Their local area network is operated under the MS-DOS environment and involves the use of various protocols necessary for communications with ARPANET.

The aim of this thesis is to provide an efficient data communications environment limited to the local area network of Z-100 workstations running under the CP/M-86 operating system.

B. PROJECT DESCRIPTION

1. Proposed Capabilities

a. Local File Transfer

Any microcomputer should be able to transfer a file to any other microcomputer asynchronously. Additionally, any microcomputer should be able to 'broadcast' files to multiple microcomputers simultaneously.

b. Local Message Transfer

Any microcomputer should be able to transfer messages in the same manner as files, with the additional capability of having one microcomputer serve as a 'bulletin board' for the others.

c. Directory Transfer

Any microcomputer should be able to transfer a directory in the same manner as messages and files.

d. Online User Identification

Any microcomputer should be able to obtain a 'net status'; the identities of all currently active workstations in the network, at any given time.

2. Target Hardware

The proposed local area network consists of up to twenty four Z-100 microcomputer workstations connected via RS-232 communications ports to a central MULTIBUS based single board computer, which acts as a central switchboard to provide communications between the workstations.

3. Software

All the applications software for the microcomputers and the processing software for the single board computer acting as a switchboard has been written by the author for this thesis.

C. STRUCTURE OF THE THESIS

The majority of this thesis is the program code that implements the network. The accompanying text provides system description, design decisions, problems encountered, and operating and maintenance procedures. Chapter II describes the hardware of the system, Chapter III details the 8086 CPU, and Chapter IV explains the CP/M-86 operating system. A description of the JANUS/Ada language detailing useful

features, problems encountered, and lessons learned from the prospective of a newcomer to the language with reference to the full ADA language is provided in Chapter V. Chapter VI explains the design methodology and the details of the software implementation with descriptions of each major module.

The appendices provide the program code, user and maintenance manuals, a glossary, and a bibliography.

II. HARDWARE

A. THE 86/12A SBC

The 86/12A single board computer is a complete computer system on a single printed circuit board. It includes a 16 bit 8086 CPU, 32K, expandable to 64K, bytes of dynamic RAM, a serial communications interface, three programmable parallel I/O ports, programmable timers, priority interrupt control, MULTIBUS interface control logic, bus expansion drivers for interface with other compatible MULTIBUS expansion boards, and up to 16K bytes of ROM. Table 2.1 lists the possible I/O port addressing asignments.

TABLE 2.1

86/12A I/O ASSIGNMENTS

I/O Address	IC	Function
00C0	8259A	write: ICW1, OCW2 & OCW3
or 00C4	Programmable Interrupt	read: Status and Poll
00C2	Controller	write: ICW2, ICW3, ICW4, OCW1
or 00C6		read: OCW1 (Mask)
00C8	8255A	write: port A (J1)
	Programmable	read: port A (J1)
00CA	Peripheral	write: port B (J1)
	Interface	read: port B (J1)
00CC		write: port C (J1)
		read: port C (J1), Stat
00CF		write: Control
		read: none

00D0	8253 Programmable Interval Timer	write: Counter 0 (load cnt/N) read: Counter 0
00D2		write: Counter 1 (load cnt/N) read: Counter 1
00D4		write: Counter 2 (load cnt/N) read: Counter 2
00D6		write: Control read: none
00D8 or 00DC	8251A USART	write: Data (J2) read: Data (J2)
00DA or 00DE		write: Mode/Command read: Status

The 8538 Eight Channel Communication Expansion Board is a fully programmable synchronous/asynchronous serial communication channel with RS232C interfaces connected to IC2651 USARTs for serial communications with other devices, and is compatible with the MULTIBUS system. The total address space for each board is 64, or 40 Hex, locations. Each board has a base address that is selectable by DIP switches on the board and is set to 0100 Hex for this implementation. Disregarding the base address for the moment, board 1 would start at 0, board 2 at 40 Hex, and so on. The primary locations of interest are the data register, the status register, the mode register, and the command register. These locations are 0-3 for port 0, 4-7 for port 1, and so on up to 20 Hex for the

eight USARTs. The remaining address locations are for interrupt handling and are not used in this implementation. The port addressing is shown in Table 2.2.

TABLE 2.2

USART ADDRESSING

Address (Hex)	Function			
	R/W: Data	R: Status W: Sync1/Sync2/Dle	R/W: Mode	R/w: Command
	"	"	"	"
0-3	"	"	"	"
4-7	"	"	"	"
8-B	"	"	"	"
C-F	"	"	"	"
10-13	"	"	"	"
14-17	"	"	"	"
18-1B	"	"	"	"
1C-1F	"	"	"	"
20,28,30,38	Port reset register (write only)			
21,29,31,39	N/A			
22,2A.32,3A	Transmit interrupt register			
23,2B,33,3B	Transmit interrupt requests			
24,2C,34,3C	Transmit interrupt mask			
25,2D,35,3D	Transmit interrupt requests			
26,2F,36,3E	Ring detects			
27,2F,37,3F	N/A			

B. THE Z-100

The Zenith Z-100 is a dual processor 8085/8088 computer with either one or two 5.25 inch floppy disk drives and one Winchester hard disk drive providing up to 750K bytes of directly addressable RAM. It is powered by a 5MHZ clock.

The Z-100's main circuit board contains the 8085 and 8088 CPUs and the S-100 IEEE 696 bus. It has the capacity for 192K bytes of memory. Also contained on the main board is

the 8041A keyboard processor, two RS-232 serial interfaces and connectors, one parallel interface and connector, and the video circuit board interface.

The video circuit board contains the CRT controller and supports a bit mapped video system with up to three banks of 32K byte memory devices for red, green, and blue.

The floppy disk drive controller, which can support up to four 5.25 inch and four eight inch drives, and the Winchester disk controller are on separate cards and are each connected to one of the slots of the S-100 bus.

Other hardware features of interest include: the 8259A Programmable Interrupt Controller, the 68A21 Peripheral Interface Adapter, the 8253 Programmable Interval Timer, and the 2661 Enhanced Programmable Communications Interface. The Z-100 has two serial ports, labeled J1, the printer port, and J2, the modem port, both of which are connected through the 2661 interface.

III. THE CENTRAL PROCESSING UNITS

A. THE 8086

The 86/12A single board computer uses the INTEL 8086 microprocessor for its CPU (Central Processing Unit). The 8086 is a high performance, general purpose 16 bit microprocessor. It has a 20 bit address bus, allowing access to a full megabyte of memory. Since the largest register in the 8086 is only 16 bits, it uses segmentation to form 20 bit addressess from the four 16 bit address registers; the CS (Code Segment), the DS (Data Segment), the SS (Stack Segment), and the ES (Extra Segment). These four registers reside in the BIU (Bus Interface Unit).

The EU (Execution Unit) contains nine 16 bit registers interfaced to a 16 bit data bus, four of which are byte or word addressable; the AX (AH, AL), BX, CX, and the DX, and four of which are only word addressable; the SP (Stack Pointer), the BP (Base Pointer), the SI (Source Index), and the DI (Destination Index). The remaining register is the flag register which has nine usable bits; Carry, Parity, Auxiliary Carry, Zero, Sign, Trap, Interrupt, Direction, and Overflow. Lastly, there is one 16 bit IP (Instruction Pointer) register.

The BIU and the EU operate asynchronously in the 8086. Additionally the BIU has an instruction object code queue. These two features combine to virtually eliminate instruction fetch time. By fetching the next instruction, while the

previous is being decoded and executed, and loading it in a queue, the 8086 ensures that there is always an instruction ready the instant it is required. Only when a jump instruction causes the current instruction sequence to be changed, is the instruction queue flushed.

B. THE 8088

The Z-100 uses both the 8 bit 8085 CPU and the 16 bit 8088 CPU. The 8088 is the processor of interest to this thesis. The 8088 is compatible with the 8086 and is (to the programmer) virtually identical. In particular, the 8088 programmable registers and addressing modes are exactly the same. The significant difference is that it has an 8 bit data bus versus the 16 bit data bus of the 8086.

The differences between the two CPUs are evidenced in execution times. The 8088 has a four byte instruction queue compared to the six byte queue in 8086. This results in more code fetches per instruction and slows down instruction processing. The 8 bit data bus requires the 8088 to take two bus cycles whenever the 8086 would have used only one to fetch 16 bits of data. All other execution time values are identical.

IV. THE OPERATING SYSTEM

CP/M-86 is an operating system designed by Digital Research for the 8086 and 8088 sixteen bit microprocessor. It contains three program modules: the CCP, the BDOS, and the BIOS. Entry to the BDOS is provided through the reserved software interrupt #224 (E0 Hex), while entry to the BIOS is provided by either a jump vector located at offset 2500 Hex from the operating system base, or by use of the BDOS function #50. For this implementation, only entry to the BDOS was required. BDOS functions were used for keyboard input, console output, and file operations. Table 4.1 lists each function call used with entry and return parameters indicated.

Access to files in CP/M-86 is achieved by use of the File Control Block (FCB) whose format is shown in Table 4.2. Each FCB is identified by specifying its relative offset from the data segment register. When reading from or writing to a file, BDOS uses sequential 128 byte records to transfer information from the file into memory at the current Direct Memory Address (DMA) and vice-versa. The DMA can be specified by the user as the relative offset from a specified or default DMA base.

CP/M-86 uses one directory entry per 16K bytes of file data, termed an extent. During sequential reads and writes the "cr" field of the FCB is incremented for each 128 byte record until the next extent is required. Unless no more

TABLE 4.1

BDOS FUNCTIONS

ENTRY =>	FUNCTION =>	RETURN
CL: 02H, DL: Ascii Char	Console Output	None
CL: 06H, DL: 0FFH (Input); 0FFH (Status); Char (Output)	Direct Console I/O	AL: Char, Status
CL: 0DH	Reset Disk System	None
CL: 0FH, DL: Selected Disk	Select Disk	None
CL: 0FH, DX: FCB Offset	Open File	AL: Return Code (0-3, 0FFH)
CL: 10H, DX: FCB Offset	Close File	AL: Return Code (0-3, 0FFH)
CL: 11H, DX: FCB Offset	Search For First	AL: Directory Code
CL: 12H,	Search For Next	AL: Directory Code
CL: 13H, DX: FCB Offset	Delete File	AL: Return Code (0, 0FFH)
CL: 14H, DX: FCB Offset	Read Sequential	AL: Return Code (0, 1)
CL: 15H, DX: FCB Offset	Write Sequential	AL: Return Code (0, 1, 2)
CL: 16H, DX: FCB Offset	Make File	AL: Return Code (0, 1, 2, 3, 0FFH)
CL: 1AH, DX: DMA Offset	Set DMA Address	None

TABLE 4.2

FCB FORMAT

dr	f1	f2	/	f8	t1	t2	t3	ex	s1	s2	rc	d0	/	dn	cr	r0	r1	r2
00	01	02	...	08	09	10	11	12	13	14	15	16	...	31	32	33	34	35

where;

dr:	drive code (0 - 16) 0 => default drive 1 => drive A 2 => drive B ... 16 => drive P
f1...f8:	File name in Ascii upper case
t1 - t3:	File type in Ascii upper case (t1 high bit = 1 => R/O file) (t2 high bit = 1 => Sys file, no DIR)
ex:	Current extent number (0 - 31) Set to 0 by user before file I/O
s1:	Internal use
s1:	Internal use, set to 0 on Make, Open, and Search
rc:	Record count for ex (0 - 128)
d0...dn:	CP/M-86 internal use
cr:	Current record for Read/Write Sequential Set to 0 by user
r0 - r2:	Optional random record number

record or directory space exists, the next extent is automatically opened. Then the "cr" field is reset to zero and a new directory entry for the new extent is created.

When directory searches are made, the 128 byte record containing the matched entry is placed in memory at the DMA address. For files with multiple extents, the first directory entry is matched. There are four entries per record.

The location of the matched entry is computed by multiplying the offset returned by the function call by 32 and adding it to the DMA address.

DDT86 proved to be an invaluable tool to the development of this thesis. DDT86, a utility program supplied with CP/M-86, is a dynamic, interactive debug and test program for the CP/M-86 environment. It worked excellently with both the high level JANUS/Ada code and the assembly language subroutines interfaced with the high level code. DISASM86, the disassembler provided with the JANUS/Ada system, presented the high level code in its post-compiled assembly form which could then be debugged using DDT86. Completion of this thesis would not have been possible without the use of DDT86. Its only drawback was a tendency to 'slow' down a program to the point where it would work under DDT86, but would fail in the real time environment outside DDT86, which caused understandable confusion.

Additional detailed information about CP/M-86 can be found in [Refs. 2 & 3].

V. THE PROGRAMMING LANGUAGE

ADA is the Department of Defense's mandated language for embedded computer systems. According to DOD directive 5000.31 [Ref. 4], "all mission critical defense systems that enter advanced development status after 1 Jan 84, or that enter full scale development after 1 Jul 84" will be coded in ADA. Introduced in its final form in September 1980, ADA has inspired much activity, discussion, and debate. Due to its large size and complexity, commercial compilers are only now becoming available.

JANUS/Ada, an implementation designed for microcomputers, was chosen for this project because of its availability, and its suitability to systems programming tasks. Although not fully ADA compatible in its treatment of strings, use of inline ASM statements, or the data type Byte, and lacking the capability for exception handling or concurrent tasking, research in the JANUS/Ada implementation should prove invaluable to the effective use of the full language itself; a language destined to be the Department of Defense's standard for embedded systems.

This author's programming experience prior to this project was limited to only the basics of Pascal and Assembly languages. According to Bernard [Ref. 5], my experiences can best be described by:

While people who have a high level language should be able to learn what is referred to as 'the Pascal subset' of ADA with about four weeks of full-time study, many educators report that it takes six months to learn to make effective use of the language.

Also required are a firm knowledge of structured programming and software engineering. It is clear, however, that the ADA designers were correct to base the language syntax on Pascal, a language designed and recognized as the standard to teach students about structured programming.

The most impressive and useful control structure in ADA is the 'Loop' statement. The ability to exit a loop wherever and whenever desired, combined with the traditional constructs of 'For' and 'While', makes the Loop statement applicable to an almost infinite set of design structures and truly supports the concept of structured, easy to read programming. This feature alone separates ADA's control structures from Pascal, Cobol, Fortran, and others.

JANUS/Ada's resident assembly language interface is also extremely useful. Routines that require high speed and efficiency or specific bit/byte manipulation could be coded in assembly language and called from the high level program. This feature answers the traditional systems programmer's need for control of, and access to, the specific machine. The ease with which assembly code could be used in conjunction with high level JANUS/Ada code was very important to this project. Almost all of the I/O and file operations required in the programs were coded in assembly language. This was

partly for speed and efficiency and partly because many of the I/O and file operations built into JANUS/Ada would not work (for reasons undiscovered) after a single call to the CP/M-86 operating system was issued.

ADA's use of packages as the unit of program modularity is an important step in the evolution of structured programming languages. Packages contain data, procedures, and/or functions that for reasons of a particular program's modular design are grouped together to form a unit. A program's various packages are then linked together for execution. Packages support efficient modular design and allow for the effective construction of program libraries of frequently used routines and data structures for programmers to use. Data and routines are made available to users and other packages through Specifications, which provide all the necessary information for their use while hiding the details of the implementation, thus supporting the principle of information hiding.

Packages are also the main instrument of separate compilation, which is becoming a requisite feature of modern programming languages. Unless the vehicle for communication between linked packages, the specification, is changed, modifying a single package requires only its recompilation and not the recompilation of packages referencing it or referenced by it.

Separate compilation is particularly important in JANUS/Ada, and by extension, ADA, because the size and complexity of the language dictates that compilers for it be also large, complex, and relatively slow. This causes software development problems of time and storage.

The compiler itself, while a bit slow, provided helpful compile and run time error checking. Because ADA (and JANUS/Ada) is a strongly typed language, an efficient compiler should be able to detect a majority of errors at compile time, thereby saving development time. The JANUS/Ada compiler does so and even corrected certain syntax errors itself, producing a working program. While it is, in general, difficult to produce simple yet completely explanatory error messages, the JANUS/Ada compiler does a noteworthy job.

[Ref. 6], provided with the JANUS/Ada system, was more than adequate overall. It was not sufficient (and admitted so) to teach the JANUS/Ada language, but its manuals and appendices on the compiler, the assembler, the linker, etc. were first rate. Its description of library packages was also very helpful. [Ref. 7] provided all the necessary information to begin learning ADA in a very practical and concise manner.

VI. THE IMPLEMENTATION

A. METHODOLOGY

This author's inexperience with the hardware, the programming language, and software development of this nature (systems programming) and this scope (larger by far than anything attempted before), necessitated complete decomposition and modularization of the problem. The principle of top-down programming was used as the problem was broken into smaller and smaller functional elements until the most basic functions were identified. Such functions as basic input and output, opening and closing files, and interfacing assembly language with JANUS/Ada were coded and tested. These functional entities were combined in a building block manner, until larger and more complex functions such as character transfer and file transfer between two Z-100 were completed. Eventually, interface with the concentrator was achieved as the system began to take shape. The system was finished when the highest levels of the program hierarchy were finally completed and all the decomposed modules of the problem were coded, tested, and linked.

B. SYSTEMS DESIGN

The problem of designing a star cluster local area network may be decomposed into two parts:

1. Programming the Concentrator.
2. Programming the Z-100.

1. The Concentrator

The role of the concentrator is to process network commands from the Z-100s, establish and maintain data communication connections, and route commands and data between Z-100s. The network principle of 'circuit switching' is implemented to route the commands and data. The Concentrator establishes and maintains data communication connections between the source and destination Z-100(s) until the entire user process or 'circuit' is completed.

Within the Concentrator the problem may be further decomposed into two parts. The Concentrator must poll all the ports in the network upon initiation and after each connection is terminated, to determine if any Z-100s are ready to participate in the network, and it must establish and maintain data communication connections.

Currently the Concentrator is equipped with three 8538 BLC expansion boards for a total of 24 available ports. Network expansion can be accomplished by adding more expansion boards and changing the value of the constants 'machno' and 'boardno' accordingly.

The Concentrator polls each port looking for the 'active' signal from a Z-100. If there is no response after a finite period of time, the Concentrator moves on to the next port. If the 'active' signal is received, the Concentrator stops and determines if there are processes waiting for this Z-100 from other Z-100s. If there are no

processes waiting, the Concentrator decodes the subsequent command information from the polled Z-100, and establishes the data communication connection for the requested process.

If there are processes waiting for the Z-100 being polled, the Concentrator decodes the first process in line, contacts the sender of that waiting process, and establishes a data communication connection for the transferral of the waiting process. Upon transfer of the first waiting process, or if the sender is no longer active, the waiting process is destroyed and the Concentrator returns to the polled Z-100 in order to decode the subsequent command information, and establish the data communication connection for the original requested process.

The Concentrator will satisfy at most one waiting process per polling cycle; a polling cycle is one loop around the network of twenty four ports. If there is more than one process waiting in the queue for a particular Z-100, the one which is first in the queue (first come, first served) is serviced during one polling cycle. The next queue element is serviced during the next polling cycle and so on until there are no more waiting processes.

There are two methods of transferring commands and data in this design:

1. Direct Z-100 - to - Z-100.
2. Broadcast to all active Z-100s in the network.

The Z-100 transmits to the Concentrator the transferring method and the Concentrator uses that method for all file, message, or directory transferral.

Once a data communication connection is established, the Concentrator acts as a transparent pathway, transferring commands and data without regard, and looking only for a sequence of four end of process codes (0F1h) from the destination Z-100(s). Receiving this sequence, the Concentrator terminates the connection and resumes polling where it left off. The Concentrator polls in an endless loop, stopping only to process requests from the ports of the network.

For single transferrals, error checking is performed by the sending and receiving terminals. For broadcast transferrals, the Concentrator performs the error checking between itself and the receiving terminals while the sender performs error checking between itself and the Concentrator.

If a data communication connection cannot be established, the record data structure 'connection' containing fields for source, destination, and process is queued in a FIFO queue implemented as an array of records. The queue is limited to one process per source Z-100 per destination Z-100 and is limited to message and file processes. It is this queue that the Concentrator checks after receiving the 'active' signal from any Z-100 during the polling process.

2. The Z-100

The role of the Z-100 is to communicate with the user and the Concentrator for the purposes of processing network application programs. Commands are transmitted from the user to the Concentrator, enabling the establishment of the proper data communication connection. Then the source Z-100 transmits commands and data to the destination Z-100(s).

The network applications consist of three Z-100 to Z-100 functions; file transfer, message transfer, and directory transfer, and one Z-100 to Concentrator function; net status. Within each Z-100 to Z-100 function is a sending process and a receiving process. It is these processes that serve as the network unit of execution. The source Z-100 initiates a process, the Concentrator establishes a connection with the destination Z-100(s), and maintains the connection until the process is finished. Then the next process is initiated and so on.

Both the Z-100 and the Concentrator use the record data structure 'connection' to establish data communication connections. 'Connection' contains three fields; 'source', 'destination', and 'process'.

```
TYPE conctn IS
   RECORD
      source: byte;
      destination: byte;
      process: byte;
   END RECORD;
```

The source Z-100 creates the connection record and transmits it to the Concentrator where it is decoded and the proper pathway is established. When the destination Z-100(s) acknowledge the connection, communication begins. If the connection can not be established, the connection record is stored by the Concentrator in the queue. Upon completion of the process, the connection record in the Concentrator and the Z-100 is destroyed.

The Z-100 network application programs are menu driven and implemented in an infinite loop. The Z-100 transmits an 'active' signal to the Concentrator and waits to be polled by the Concentrator. When polled the user is told if there is a process waiting for him/her and is requested to receive that process. Then the user selects a process from the menu and the Z-100 and the Concentrator carry out that process as described above. After process completion, control is returned to the infinite loop which transmits an 'active' signal again, waits to be polled, asks for user input, and so on until the user terminates the loop with "control x".

3. Command and Data Communication

A main concern of this thesis is efficient data communications. Fast, error free transmission of commands and data along the data communication connection for both point to point and broadcast, is achieved by an immediate echo method implemented in JANUS/Assembler language for

CP/M-86. The method uses the fact that instruction execution time is much shorter than data transmission time and attempts to minimize wait times as much as possible. All bytes received by any procedure are immediately echoed and the received echoes are then checked for error.

The sending procedure transmits a byte and then waits for an echo from the receiver. It compares the echo with the transmitted byte. If an error is detected, the sending procedure, depending on its function, either retransmits immediately or transmits an error code followed by the retransmitted byte. (For the broadcast method the byte is transmitted to all destination Z-100s before the echoes are received and compared.)

The receiving procedure receives a byte and then immediately echoes it before proceeding to process the received byte. If an error code is received when it returns to receive the next byte, the receiver erases the byte in error and returns to receive the retransmitted byte.

The method is sufficiently general in design to handle more difficult errors than just errors in data bytes transmitted from the sender to the receiver. For instance, the method will detect and correct errors in the echo transmitted back to the sender. This is a particularly insidious error because the receiver actually received the correct data byte, but the sender doesn't think so and sends out an error code or retransmits the data byte. The method

will also correct multiple errors such as when a data byte is transmitted in error and so is the subsequent error code. The sender will stay with the error until it is corrected to its satisfaction and the receiver will always know how many received data bytes are in error and how many to erase. This is true even when a procedure is both a sender and a receiver as in the procedure 'Setup'. Additionally, every receiving procedure has a finite waiting period to ensure the final byte and echo are received properly.

4. Design Considerations

a. Assembly Language

JANUS/Assembler combined with CP/M-86 BDOS function calls was used for all I/O subroutines except for the display of non-conditional strings to the user. This was decided because of the need for speed, efficiency, and implementation of the data communication connection. Because of an undiscovered bug in JANUS/Ada, use of CP/M-86 function calls invalidated the perfectly adequate built in JANUS/Ada file operation routines. Thus these functions too were implemented in JANUS/Assembler with CP/M-86 BDOS function calls.

b. Memory Management

The Concentrator programs are estimated to occupy 25K of the 64K bytes of memory in the Concentrator. Due to the limited size of the queue for waiting processes, and to the use of message switching as the principle of network

communication, circuit management was deemed not necessary and was not implemented.

c. Process Coordination

The network processes do not use interrupts for coordination. Processes execute sequentially until communication is required. This communication is designed such that when, say, a sender is waiting to transmit specific commands or data, the receiver will eventually 'rendezvous' (although not in the concurrent sense) and the proper receiving procedure will receive and process the transmitted commands or data. When necessary, such as when polling ports, processes will wait a finite period of time and then move on. Occassionally, a process will stop until the receiver tells it to continue. Blocks of data are identified by end of block codes (ØFFh), and codes are transmitted to tell the receiver whether or not this was the last block. Special care was exercised in the implementation to ensure that deadlock did not occur ie: both processes expecting to receive or to transmit.

The Concentrator, which is both a receiver and a sender, employs the same design principles when polling ports, checking or updating queues, and establishing data communication connections. Once a connection is established, it treats everything as a block of data, moving on only when a sequence of four end of process codes (ØF1h) from the destination Z-100(s) is received.

d. The Data Type Byte

Use of the JANUS/Ada predefined data type Byte, which is not ADA standard, was necessitated by the network requirement to transmit all eight bits of each data byte and the method used to transmit and receive commands and data. The data type Character with only seven bits, and the data type Integer, which is not well suited for the manipulations needed for transmission, were not sufficient to represent the data byte used in command and data communication in this implementation.

C. SYSTEM EXECUTION

The implemented network application programs are menu and command driven. At program start and after each process, the 'active' signal is transmitted to the Concentrator and the program waits to be polled. When polled the program is informed if there are processes waiting for this Z-100. If there are processes waiting, the user is asked to receive them. Then the user is presented with the following menu:

```
CTRL_S = SEND FILE
CTRL_R = RECEIVE FILE
CTRL_T = SEND MESSAGE (TALK)
CTRL_L = RECEIVE MESSAGE (LISTEN)
CTRL_P = SEND DIRECTORY
CTRL_G = RECEIVE DIRECTORY
CTRL_W = GET NET STATUS (WHO)
CTRL_B = RECEIVE BULLETIN BOARD
OTHERS = RECHECK FOR INCOMING FILE OR MESSAGE
```

The user makes his or her selection and is placed in the appropriate environment. Upon completion of each process,

the user can press any key and be returned to the main menu or pause indefinitely. Upon return to the main menu, the user is given the option of exitting using "CTRL_X" or continuing "ANY KEY".

To send a file, the user is prompted for the disk drive, file name, and file type of the file he wishes to send. The user's input is displayed as he types and invalid entries (those characters not permitted in file names and file types by CP/M-86) are flagged. A sample entry might look like this:

```
FN.FT: myfil? INVALID ENTRY
myfil_
```

Once entered the file name and type are capitalized in accordance with CP/M-86 format and the user's choice is displayed for confirmation:

```
C: MYFILE.TXT IS SELECTED. PRESS RETURN TO CONFIRM,
ANY OTHER KEY TO RESELECT.
```

Upon confirmation, the user is prompted for the destination terminal (or broadcast) and the source terminal. Then a search is made for the file. If it can not be found an error message is displayed and the user is given the opportunity to reselect. If it is found the file is opened. Then, if the destination is active, the transmission begins. Otherwise, a message indicating that the destination was inactive is displayed and the user is returned to the main menu.

The file is sent in succesive 128 byte data blocks until end of file is reached. After each data block, the sender

indicates to the receiver whether or not end of file has occurred and stops until the receiver tells the sender to continue. Once the file is sent, it is closed and the user is returned to the main menu.

To receive a file, the user is prompted as above for drive, file name, and file type. Upon confirmation, the file named by the user is first deleted and then created. This is done to ensure no errors occur during the create operation and has the effect of overwriting the file if the user chooses a file already on the disk. After creating the file, reception begins.

The file is received in 128 byte data blocks and after each block is written to disk, end of file is checked and the sender is told to continue. Once the file is received, it is closed and the user is informed of the number of bytes received. Error messages are displayed if there is no more disk space or directory space on the receiver. The user is then returned to the main menu.

To send a message, the user is prompted to begin typing and told that the maximum message length is 1600 characters. All characters entered except "CTRL_Z" are sent. The message is organized in a 'page' format of twenty lines with 80 characters each. A limited correction capability is included, consisting of typing "backspace" or "CTRL_H" and then retyping the character. However, the error, the backspace or CTRL_H, and the retyped character all count as typed

characters. The message format consists of twenty lines of 80 'keystrokes' each. Users must keep this in mind when typing.

"CTRL_Z" stops message typing and "CTRL_S" sends it. Any other key allows the user to retype the message. After "CTRL_S", the user is prompted for the source and destination (or broadcast) terminals as above. If the destination is active, the message is transmitted as one data block with appropriate end of message codes. The user is returned to the main menu upon completion of the transmission.

To receive a message, the user types "CTRL_L" from the main menu and reception occurs. The message is displayed and the user is informed of end of message and the number of bytes received. The user is then returned to the main menu.

To send a directory, the user is prompted for the disk drive for which directory he wishes to send. (Presumably in response to a request.) After confirmation, the user is prompted for the source and destination (or broadcast) terminals as above. The directory of the selected drive is then searched for the CP/M-86 wild card file name and type (????????.???) causing the entire directory to be matched. Each directory entry is placed in the transmission data block until there are eight directory entries for a total of 128 bytes. If the destination is active this data block is then transmitted , preceded by the disk drive, with a code indicating whether or not end of directory has occurred. If there

are no files on the selected disk a "NO FILES ON SELECTED DISK." string is transmitted to the receiver. The user is returned to the main menu upon completion of the transmission.

To receive a directory, the user is prompted as above for drive, file name, and file type in which he wishes to store the incoming directory. Upon confirmation, the file named by the user is first deleted and then created. This is done to ensure no errors occur during the create operation and has the effect of overwriting the file if the user chooses a file already on the disk. After creating the file, reception begins.

The drive of the directory is received first and then the first 128 byte data block. After checking for end of directory, the eight directory entries of the data block are displayed and the user is asked if he wishes to save them on file. Each data block is received the same way. Those data blocks the user wishes to save are written to disk and the rest are discarded. Error messages are displayed if there is no more disk space or directory space on the receiver. When end of directory is reached the reception is ceased, the file closed, and the user returned to the main menu. The file created during this operation should be edited by the user as it may contain characters from each directory entry that are not printable.

To obtain a current net status of the active terminals in the network, the user is prompted for the source and destination terminals as above with the exception that he must enter his terminal for both source and destination. Upon confirmation, a list by number of all active terminals is displayed and the user is returned to the main menu.

Messages are sent the bulletin board, if it is active, by selecting machine #24 when prompted. The bulletin board is 'read' by selecting CTRL_B at the main menu. If the bulletin board is active, all the current messages are transmitted, in 128 byte blocks, and stored in a user specified file, chosen in the same manner as the receiving of file transfers. This file can then be perused at leisure. The bulletin board can hold up to twenty messages of 1600 bytes each. The twenty-first message replaces the first message and so on.

VII. CONCLUSIONS

The research objectives of this thesis; coding in JANUS/Ada, allowing single or multiple transfers of files, messages, and directories, and sharing of local resources in a laboratory environment, were satisfactorily completed. However, the complete testing and demonstration of the network in operation was not achieved.

The utilities resident in the Z-100 workstations were completely and satisfactorily tested. So too were the establishment of data communications connections and data transfer through the concentrator of files, messages, and directorys from a single Z-100 to another single Z-100 and to multiple (two) Z-100s. The polling mechanism in the concentrator, obtaining net status, and the storing of messages in and retrieval from the bulletin board were also satisfactorily tested.

Remaining to be tested are: the mechanism for the storing of processes for which a connection could not be established and the mechanism for placing and servicing the waiting processes in the queue. Finally, the testing of the entire system of 23 Z-100 workstations and one Bulletin board under all conditions and loads awaits demonstration.

In sum, each module and subroutine was separately compiled and tested, functional integration of parts of the

system were tested and satisfactorily demonstrated, but the complete integration of the entire network was not accomplished.

To achieve final system integration and completion of the network, it is recommended that a two step testing process be conducted. First, set up, under laboratory conditions, a small network consisting of seven Z-100 workstations and a bulletin board using one full 8538 expansion board. In this set up, fully test the package 'Poll' in various experimental situations to ensure that it handles all circumstances as designed and that the queue operations function properly. Second, establish the full network and test it in an operational environment under all conditions of load and function.

The experience of coding this thesis was very challenging and stimulating. Despite my very limited programming background in Pascal, I was able to quickly grasp the fundamentals of Janus/Ada and begin coding small test procedures. As the requirements for the project became clearer, coding intensified and increased in size and scope.

However, I found myself habitually using Pascal or Pascal-like structures. I was tapping very little of the power and richness of Janus/Ada. For instance, I would use

```
IF condition THEN
   EXIT;
END  IF;
```

rather than

```
EXIT WHEN condition;
```

to exit loops. Features such as attributes, subtypes, derived types, and ranges were used very little in the beginning. Additionally, the nature of the project precluded working with real or fixed point types and all their features plus many of the built in or provided library subroutines.

It was in the use of the LOOP statement that I began to explore the full power of Janus/Ada over Pascal. I quickly learned to tailor the loop statement to achieve any effect I desired and found it extremely useful in structuring my programs. Gradually, in this and other structures, I began to use Janus/Ada more elegantly and to remove 'Pascal' from my coding, although some always remained.

Surprisingly, the grouping of programs, subroutines, and data into packages and the linking together of these packages into a single program proved very easy to assimilate. The understanding and use of the scope and visibility rules, however, was harder to grasp. As a result, the use of selected components for naming variables in one package while in another package, or the use of private types were not implemented.

The first real problems occurred in the use of input and output, usually the hardest part of any language. For

reasons previously explained, assembly language subroutines interfaced with Janus/Ada programs were used for virtually all input and output. My previous assembly language experience was with 8080 mnemonics, but the use of 8086 mnemonics was easy to learn because I was required to use only those instructions that had counterparts in 8080 and I only had to learn the new register and addressing schemes. However, the interface with Janus/Ada proved extremely difficult to master. It took much trial and error to realize exactly how the parameters were passed and returned from the assembly language subroutine and how the stack was manipulated when the subroutine was called.

Lastly, the requirements of data communication were the hardest to implement. The idea of two programs executing on two different machines and communicating with each other was entirely new. The errors that resulted were difficult to diagnose and correct. In my previous experience, programs very rarely encountered infinite loops; either they didn't work correctly or they produced wrong answers, but they always finished. In data communication, infinite loops were frequent as the programs deadlocked waiting for the other to stop or start communicating. The synchronization of procedures and routines in programs executing in parallel, so that procedures that needed to communicate would 'meet' at the right place and the right time, was the most difficult aspect of coding this thesis.

This thesis demonstrated the following capabilities. First, the ability of JANUS/Ada, and by extension, ADA, to effectively and efficiently perform complex data communication and network functions. Second, the viability of clustering microcomputers for the sharing of resources and the enhancement of data comunications and transfer. Finally, the ability of an inexperienced programmer to learn ADA and use it in the solving of complex computer problems.

APPENDIX A

USER'S MANUAL

A. GETTING STARTED

The system is designed to operate on the Zenith model 100 microcomputer connected to the concentrator in the NPS microcomputer lab. The file Xfermain.cmd should be on the hard disk and the CP/M-86 operating system should be selected.

B. SYSTEMS OPERATION

The command 'xfermain' will place the user in the network. The first message should be:

'WAITING TO BE POLLED.'

Note: System messages are always in uppercase.
This means that this Z-100 is waiting to be recognized by the network. A wait of some time may result if the system is busy. If the wait becomes excessive, see the system's maintenance personnel. When recognized, the next message should be:

'CHECKING FOR INCOMING FILE OR MESSAGE.'

If there is an incoming file or message, the user will be asked to receive it prior to continuing with the user's request.

After the incoming data has been received or if there is no data, the following menu will be displayed:

```
CTRL_S = SEND FILE
CTRL_R = RECEIVE FILE
CTRL_T = SEND MESSAGE (TALK)
CTRL_L = RECEIVE MESSAGE (LISTEN)
CTRL_P = SEND DIRECTORY
CTRL_G = RECEIVE DIRECTORY
CTRL_W = GET NET STATUS (WHO)
CTRL_B = RECEIVE BULLETIN BOARD
OTHERS = RECHECK FOR INCOMING FILE OR MESSAGE
```

Selecting the proper entry will place the user in the desired environment.

C. SENDFILE

Once in Sendfile, the command messages are self explanatory. The program forces the input of a CP/M-86 acceptable file name and file type, but the user must ensure that the file exists on the selected disk. An error message will result otherwise. The next step is selecting the destination machine (numbered Ø1 through 23) or broadcast to all (number ØØ). If the requested destination is not presently in the network, the request to send a file will be queued along with the intended destination. Some delay may occur in the sending of a file if the file is very large or the network is very busy. Excessive delay should be referred to the systems maintenance personnel.

D. RECEIVEFILE

Once in Receivefile, the command messages are self explanatory. The program prompts for a file name and file type for the incoming data. It forces the input of a CP/M-86 acceptable file name and file type, but the user must ensure that sufficient space exists for the file on the selected disk. An error message will result otherwise. Some delay may occur in the receiving of a file if the file is very large or the network is very busy. Excessive delay should be referred to the systems maintenance personnel.

E. TALKING

Once in Talking, the command messages are self explanatory. Message length is 1600 characters organized in 20 lines of 80. Errors can be corrected using backspace or 'control h', but each keystroke counts towards the 1600. The next step is selecting the destination machine (numbered Ø1 through 23), broadcast to all (number ØØ), or bulletin board (number 24). If the requested destination is not presently in the network, the request to send a message will be queued along with the intended destination. Some delay may occur in the sending of a message if the message is very large or the network is very busy. Excessive delay should be referred to the systems maintenance personnel.

F. LISTENING

Once in Listening, the message is received immediately and displayed as typed.

G. WHOS_THERE

Once in Whos_there, the user is prompted to enter his machine number as the destination. Then the net status is received.

H. RECEIVE_DIR

Once in Receive_dir, the command messages are self explanatory. The program prompts for a file name and file type for the incoming data. It forces the input of a CP/M-86 acceptable file name and file type, but the user must ensure that sufficient space exists for the file on the selected disk. An error message will result otherwise. The user may save only those directory entries he desires. Some delay may occur in the receiving of a directory if the directory is very large or the network is very busy. Excessive delay should be referred to the systems maintenance personnel.

I. PRESENT_DIR

Once in Present_dir, the command messages are self explanatory. The user is prompted to select the destination machine (numbered 01 through 23) or broadcast to all (number 00). If the requested destination is not presently in the network, the request to send a directory will not be queued and is destroyed. Some delay may occur in the sending of a directory if the directory is very large or the network is very busy. Excessive delay should be referred to the systems maintenance personnel.

J. BULLBRD

Once in Bullbrd, the command messages are self explanatory. The program prompts for a file name and file type for the incoming data. It forces the input of a CP/M-86 acceptable file name and file type, but the user must ensure that sufficient space exists for the file on the selected disk. An error message will result otherwise. Some delay may occur in the receiving of messages from the bulletin board if the number of messages is very large or the network is very busy. Excessive delay should be referred to the systems maintenance personnel.

K. FURTHER ACTION

When completing any one of the above tasks, the user is returned to the main menu where he may exit using 'control x' or continue for more tasks. If the user chooses to continue the program returns to the begining, waits to be recognized again, and so on.

L. WHEN ERRORS OCCUR

The system is designed to flag most errors and allow user correction without halting execution, however should errors occur which halt the program during data communication, then the system will have to be reset. Otherwise, reseting the user terminal will allow the user to rejoin the network.

APPENDIX F

MAINTENANCE MANUAL FOR Z-100 PROGRAMS

A. XFERMAIN

1. CONFIGURATION

a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)
f. Package description:

The Xfermain package is the main program for the Z-100 workstations. It begins by informing the Concentrator that this particular terminal is active and then waiting to determine if the network has any files or messages waiting for it. Xfermain then presents the user with a menu of selected options: send/receive files, send/receive messages, send/receive directory, and obtain network status. Control code keystrokes then place the user in the desired environment. Xfermain contains an infinite loop that will perform the above functions until the user terminates the session with control x. Upon termination, Xfermain informs the Concentrator it is no longer active.

2. SUBROUTINES

a. Contained: None.
b. Called:

Active
Waiting
Outconsole
Keyin
Sendfile
Receivefile
Talking
Listening
Whos_there
Receive_dir
Present_dir
Recv_bulletin
Clearscrn
Off

3. COMMENTS

Xfermain is placed on all terminals of the network, except #24, the Bulletin board, and is invoked by typing the command "xfermain".

B. XFERFILE

1. CONFIGURATION

a. Language - JANUS/Ada

b. Compiler Version - 1.47

c. Linker Version - 1.47

d. Target Hardware - Zenith Z-100 microcomputer

e. Operating System - CP/M-86 (version 1.14)

f. Package description:

The Xferfile package controls the sending and receiving of files from the Z-100 workstations. The user is prompted for the disk drive and file name to send or receive. The package parses and capitalizes the input into eight character filenames and three character file types, in order to conform with the requirements of CP/M-86. When the input has been confirmed by the user, Xferfile either opens an existing file or creates a new one using the file control blocks (FCBs) described in Chapter IV. Files are read and sent or received and written in 128 byte blocks. The sending and receiving processes coordinate with each other and mutually come to a halt when the processes are finished. Xferfile returns to Xfermain after each instance of file transfer.

2. SUBROUTINES

a. Sendfile

(1) Type: Procedure

(2) Purpose: To send a file from one terminal to another terminal or to all active terminals in the network.

(3) Description of Parameters: None

(4) Subroutines Called:

Clearscrn
Reset_disk
Drive_select
Parse_cap
Outconsole
Keyin
Open_file
Set_DMA
Enter_machine
Set_up
Read_seq
Yes
No
Send_block
Close_file

(5) Process Description:

Sendfile controls the sending of data from a file to the receiving terminal(s). The user specifies the disk, the file name, and the file type and Send_file creates a file control block (FCB) Then the user specifies the destination terminal(s) and a connection is established. The file specified by the FCB is opened by utilyzing CP/M-86 function call

#15 and data is read from the file into a data structure located at the specified direct memory address (DMA) in sequential 128 byte records by utilizing CP/M-86 function call #20. Sendfile tells the user if either an inappropriate file name or type is used, if the file specified is already open, or if the specified file cannot be found on the specified disk. If the receiver is active, the data structure containing the 128 byte record is then transmitted. Sendfile checks after each sequential read to determine if end of file has been reached and sends a yes or a no code accordingly. After the transmission has been completed, the file is closed.

b. Receivefile

(1) Type: Procedure

(2) Purpose: To receive and store a file from a sending terminal.

(3) Description of Parameters: None

(4) Subroutines Called:

Clearscrn
Reset_disk
Drive_select
Parse_cap
Outconsole
Keyin
Delete_file
Create_file
Set_DMA
Endfile
Write_seq
Recv_block
Close_file
Put_str
Put_int

(5) Process Description:

Receivefile controls the reception of file data from a sending terminal. The user specifies the disk, the file name, and the file type to store the data and a file control block (FCB) is created. The file specified by the FCB is first deleted using CP/M-86 function call #19 to ensure no duplication occurs and then created by utilyzing function call #22. 128 bytes of data is received and stored in a data structure located at the specified direct memory address (DMA). The data is written to the file using function call #21 and the the sender is told to send the next 128 bytes of data. Receivefile checks after each block of data is received to determine if end of file has occurred. Receivefile tells the user if either an inappropriate file name or type is used, if directory space is unavailable for the initial filename, if directory space is unavailable for new extents of an existing entry, or if disk space is full. After the

reception has been completed, the file is closed and the number of bytes received is displayed.

c. Parse_cap
(1) Type: Procedure
(2) Purpose: To parse the file name and file type for invalid characters and to capitalize both to conform to CP/M-86 protocol.
(3) Description of Parameters: A value of type fcb indicating the FCB data structure is one output parameter. A value of type Integer indicating the file name length is the other output parameter.
(4) Subroutines Called:
Outconsole
Keyin
(5) Process Description:
Parse_cap controls the creation of the user specified file name and file type. The user's input is accepted and displayed. Invalid characters are flagged and the input minus the invalid character is redisplayed for continued input. Parse_cap then capitalizes the input and stores it in the appropriate fields of FCB.

C. MESSAGES
1. CONFIGURATION
a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)
f. Package description:
The Messages package controls the sending and receiving of messages from the Z-100 workstations. A maximum message size of 1600 keystrokes (including error correction) organized in 20 lines of 80 each is implemented. The user types control z to end the message and control s to send it. If the user desires to retype the message entering any character other than control s will erase the first message. After input is confirmed, the message is transmitted and received as a single block of data. The sending and receiving processes coordinate with each other and mutually come to a halt when the processes are finished. Messages returns to Xfermain after each instance of message transfer.

2. SUBROUTINES
a. Talking
(1) Type: Procedure
(2) Purpose: To send a message to one or all of the terminals currently in the network.
(3) Description of Parameters: None

(4) Subroutines Called:
Clearscrn
Outconsole
Keyin
Enter_machine
Setup
Send_block
Yes

(5) Process Description:

Talking controls the transmission of message data to receiving terminal(s). The user is prompted to begin typing the message which is displayed simultaneously exactly as typed. Talking then prompts the user for the destination terminal(s) and, after establishing connection with the receiver(s), transmits the message as a single block of data. Messages sent to destination terminal #24 are routed to the bulletin board. Talking indicates end of message by transmitting a sequence of four end of message codes.

b. Listening

(1) Type: Procedure

(2) Purpose: To receive a message from a sending terminal.

(3) Description of Parameters: None

(4) Subroutines Called:
Clearscrn
Outconsole
Keyin
Endmsg
Recv_block
Put_int

(5) Process Description:

Listening controls the reception of message data from a sending terminal. The message is received as a single block of data and terminated when a sequence of four end of message codes is received. The message is then displayed exactly as the sender typed it along with the number of bytes received.

D. DIRECTRY

1. CONFIGURATION

a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)
f. Package description:

The Directry package controls the sending and receiving of directory information from the Z-100 workstations. A FCB is created with question marks (?) in each character of the file name and type. This serves as a wild card and en-

sures that the entire directory will be sent. Directory entries are sent or received in 128 byte blocks. The sending and receiving processes coordinate with each other and mutually come to a halt when the processes are finished. Directry returns to Xfermain after each instance of directory transfer.

2. SUBROUTINES

a. Present_dir

(1) Type: Procedure

(2) Purpose: To send a directory from one terminal to another terminal or to all active terminals in the network.

(3) Description of Parameters: None

(4) Subroutines Called:

Clearscrn
Reset_disk
Drive_select
Outconsole
Keyin
Enter_machine
Set_up
Set_DMA
Driveout
Search_first
Send_string
Send_dir
Search_next
End_block
Yes
No
Put_str

(5) Process Description:

Present_dir controls sending of data from a directory to the receiving terminal(s). The user specifies the disk and a file control block (FCB) is created with the wildcard filename and type (????????.???). Then the user specifies destination terminal(s) and, if they are active, a connection is established. The directory is searched for the first match to the FCB by utilyzing CP/M-86 function call #17 and the directory information is placed into a data structure located at the specified direct memory address (DMA) in 128 byte records. An offset is computed to the unique match within this record. Present_dir transmits the 16 byte match to the receiving terminal(s). Function call #18 is used to obtain all subsequent matches. Present_dir transmits a total of eight 16 byte segments before transmitting the end of block code, so the receiving terminal(s) receive a full, contiguous 128 byte block. Additionally, Present_dir indicates end of directory by sending yes or no codes with each 128 bytes. The user is told if there are no files on a selected disk and a message stating such is sent to the receiving terminal(s).

b. Receive_dir
(1) Type: Procedure
(2) Purpose: To receive and store in a file the directory from a sending terminal.
(3) Description of Parameters: None
(4) Subroutines Called:
Clearscrn
Reset_disk
Drive_select
Parse_cap
Outconsole
Keyin
Delete_file
Create_file
Set_DMA
Endfile
Write_seq
Recv_block
Drivein
Close_file
(5) Process Description:
Receive_dir controls the reception of directory data from a sending terminal. The user specifies the disk, the file name, and the file type to store the data and a file control block (FCB) is created. The file specified by the FCB is first deleted using CP/M-86 function call #19 to ensure no duplication occurs and then created by utilyzing function call #22. 128 bytes of data is received and stored in a data structure located at the specified direct memory address (DMA). Each 128 bytes of directory data is displayed and the user is given the option of storing the information in the file created. The data is written to the file using function call #21 and the the sender is told to transmit the next 128 bytes of data. Receive_dir checks after each block of data is received to determine if end of directory has occurred. Receive_dir tells the user if either an inappropriate file name or type is used, if directory space is unavailable for the initial filename, if directory space is unavailable for new extents of an existing entry, or if disk space is full. After directory reception has been completed, the file is closed.

F. WHO
1. CONFIGURATION
a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)

f. Package description:

The Who package controls the inquiry for and reception of net status; which machines are currently active in the network.

2. SUBROUTINES

a. Whos_there

(1) Type: Procedure

(2) Purpose: To request and receive net status information.

(3) Description of Parameters: None

(4) Subroutines Called:

Clearscrn
Recv_block
Enter_machine
Setup
Put_int

(5) Process Description:

Whos_there prompts the user to enter his terminal number for both the source and destination terminals. After a connection has been established, the net status information from the Concentrator is received and displayed.

G. BULLBRD

1. CONFIGURATION

a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)
f. Package description:

The Pulltrd package controls the inquiry for receiving the current messages from the bulletin board and storing them on file.

2. SUBROUTINES

a. Recv_bulletin

(1) Type: Procedure

(2) Purpose: To receive bulletin board information.

(3) Description of Parameters: None.

(4) Subroutines Called:

Clearscrn
Enter_machine
Setup
Receivefile
Keyin

(5) Process Description:

Recv_bulletin prompts the user to enter #24 for the destination terminal. Then, if the bulletin board (terminal #24) is active, Recv_bulletin establishes a connection. It calls Receivefile which handles the actual file transfer

of the data from the bulletin board and stores it in a user specified file. Error messages are displayed if either an inappropriate file name or type is used, if directory space is unavailable for the initial filename, if directory space is unavailable for new extents of an existing entry, or if disk space is full. After reception of the bulletin board data has been completed, the file is closed.

H. BULLETIN

1. CONFIGURATION

a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)
f. Package description:

Bulletin is the package that implements the network bulletin board. It controls the reception of individual messages destined for the bulletin board and the transmission in 128 byte blocks suitable for file transfer of all current messages at a given time. Messages are received in a single block. The bulletin board has the capacity of twenty messages of 1600 bytes each. When the capacity is exceeded, the first one in is the first one out. Messages are stored as an array of records with fields for the message and its length.

2. SUBROUTINES

a. Contained: None.
b. Called:

Clearscrn
Active
Waiting
Recv_block
Endmsg
Send_block
No
Yes

3. COMMENTS

The network bulletin board is 'hard wired' into port #24. It is invoked by typing the command "bulletin". A message is displayed stating that this terminal is the network bulletin board. The program loops endlessly waiting for requests to receive messages or send current message inventory. If another terminal at another port is desired to be the network bulletin board, the package Poll must be changed to reflect the new port. (see Appendix C.) There can be only one bulletin board active at a time.

I. MYUTIL

1. CONFIGURATION

a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)
f. Package description:

The Myutil package contains various utility programs in JANUS/Ada.

2. SUBROUTINES

a. Put_int

(1) Type: Procedure
(2) Purpose: To display integers.
(3) Description of Parameters: A value of type integer to be displayed is the input parameter.
(4) Subroutines Called:
My_put
(5) Process Description:
Put_int takes the input parameter and determines its sign. Then Put_int strips off each digit, starting from the left and removing leading zeros, and displays it.

b. Clearscrn

(1) Type: Procedure
(2) Purpose: To clear the display screen
(3) Description of Parameters: None.
(4) Subroutines Called:
Outconsole
(5) Process Description:
Clearscrn outputs the CP/M-86 clear screen codes.

c. Enter_machine

(1) Type: Procedure
(2) Purpose: To get the user's input of source and destination machines.
(3) Description of Parameters: A value of type Integer indicating the source and destination machines are the output parameters.
(4) Subroutines Called:
Outconsole
Keyin
(5) Process Description:
Enter_machine prompts the user for input and then displays it for confirmation. The keyboard input is converted to Integer values which are assigned to the output parameters.

d. Drive_select
(1) Type: Procedure
(2) Purpose: To get the user's selection of disk drive.
(3) Description of Parameters: A value of type byte indicating the disk drive selection is the output parameter.
(4) Subroutines Called:
Outconsole
Keyin
Select_drive
(5) Process Description:
Drive_select prompts the user for input and then displays it for confirmation. Lower case and upper case inputs are treated the same. The appropriate parameter is then passed to the CP/M-86 function call #14, which selects the drive. The drive selection is then returned as an output parameter.

E. MYASMLIB
1. CONFIGURATION
a. Language - JANUS/Assembler
b. Compiler Version - 1.50
c. Linker Version - 1.50
d. Target Hardware - Zenith Z-100 microcomputer
e. Operating System - CP/M-86 (version 1.14)
f. Package description:
The Myasmlib package contains the library of assembly language subroutines for the Z-100 workstations.

2. COMMENTS
a. JANUS/Ada parameters for JANUS/Assembler modules are placed on the stack at subroutine call with the last parameter closest to the top and the return address on the very top. Discrete values are passed for parameters of type IN and the address of the parameter is passed for types OUT and IN OUT. Upon return to the calling program OUT and IN OUT parameters are removed from the stack along with the return address.

b. The following subroutines involve input and output to other terminals via the modem port (J2):
Send_block
Yes
No
Recv_block
Endfile
Active
Waiting
Setup
Endmsg
End_block

Send_string
Send_dir
Driveout
Drivein
Off

The method of transmission used involves an immediate echo checking procedure. The sending subroutine transmits bytes of data one at a time. The receiving subroutine receives the data bytes and echos each one immediately upon reception. The sending subroutine compares the echo with the transmitted data and checks for error. If an error is detected, the sending subroutine either simply retransmits or sends an error code to tell the receiving subroutine that the data previously received (and processed) was in error, followed by the retransmitted data. All receiving subroutines contain a finite waiting time after the last data was received to ensure that the final echo was received properly.

3. SUBROUTINES

a. Create_file

(1) Type: Procedure

(2) Purpose: To create files specified by the FCB.

(3) Description of Parameters: A value of type Integer containing the FCB address is the input parameter. A value of type Integer indicating the result of the CP/M-86 function call is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Create_file implements the CP/M-86 function call #22. It creates (and opens) the file specified by the FCB. It returns a Ø, 1, 2, or 3 if the operation was successful and 255 (ØFFh) if no more directory space is available.

b. Close_file

(1) Type: Procedure

(2) Purpose: To close files specified by the FCB.

(3) Description of Parameters: A value of type Integer containing the FCB address is the input parameter. A value of type Integer indicating the result of the CP/M-86 function call is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Close_file implements the CP/M-86 function call #16. It closes the file specified by the FCB. It returns a Ø, 1, 2, or 3 if the operation was successful and 255 (ØFFh) if the file could not be found.

c. Oper_file

(1) Type: Procedure

(2) Purpose: To open files specified by the FCB.

(3) Description of Parameters: A value of type Integer containing the FCB address is the input parameter. A value of type Integer indicating the result of the CP/M-86 function call is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Open_file implements the CP/M-86 function call #15. It opens the file specified by the FCB. It returns a 0, 1, 2, or 3 if the operation was successful and 255 (0FFh) if the file could not be found.

d. Read_seq

(1) Type: Procedure

(2) Purpose: To read data from an open file specified by the FCB.

(3) Description of Parameters: A value of type Integer containing the FCB address is the input parameter. A value of type Integer indicating the result of the CP/M-86 function call is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Read_seq implements the CP/M-86 function call #20. It reads sequential 128 byte records from an open file specified by the FCB into memory at the current DMA. It returns a 0 if the operation was successful and a 1 if no data exists at the next record position. Normally this indicates end of file.

e. Write_seq

(1) Type: Procedure

(2) Purpose: To write data to an open file specified by the FCB.

(3) Description of Parameters: A value of type Integer containing the FCB address is the input parameter. A value of type Integer indicating the result of the CP/M-86 function call is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Write_seq implements the CP/M-86 function call #21. It writes sequential 128 byte records to an open file specified by the FCB from memory at the current DMA. It returns a 0 if the operation was successful and a 1 if there is no more space in the directory for a new extent entry required when the file is larger than 16K (or a multiple of 16K), or a 2 if there is no more space on the disk for new data records.

f. Set_DMA

(1) Type: Procedure

(2) Purpose: To specify the Direct Memory Address (DMA).

(3) Description of Parameters: A value of type Integer containing the DMA address is the input parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Set_DMA implements the CP/M-86 function call #26. It sets the DMA to the value of the input parameter, which is normally the address of a specific data structure.

g. Delete_file

(1) Type: Procedure

(2) Purpose: To delete a file specified by the FCB.

(3) Description of Parameters: A value of type Integer containing the FCB address is the input parameter. A value of type Integer indicating the result of the CP/M-86 function call is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Delete_file implements the CP/M-86 function call #19. It deletes the file specified by the FCB. It returns a Ø if the operation was successful and 255 (ØFFh) if the file could not be found.

h. Select_disk

(1) Type: Procedure

(2) Purpose: To select a specified disk drive.

(3) Description of Parameters: A value of type Integer containing the selected disk drive is the input parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Select_disk implements the CP/M-86 function call #14. It designates the selected disk (Ø = A, 1 = B, etc.) as the default drive for subsequent disk operations.

i. Reset_disk

(1) Type: Procedure

(2) Purpose: To reset all disk drive systems.

(3) Description of Parameters: None.

(4) Subroutines Called: N/A.

(5) Process Description:

Reset_disk implements the CP/M-86 function call #13. It restores the file system to reset state where all drives are set to read/write and A is the default drive for subsequent disk operations.

j. Keyin

(1) Type: Procedure

(2) Purpose: To obtain input from the keyboard.

(3) Description of Parameters: A value of type Byte obtained from the keyboard is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Keyin implements the CP/M-86 function call #Ø6. It loops infinitely until a key is pressed and then the character obtained is returned as the output parameter.

k. Outconsole

(1) Type: Procedure

(2) Purpose: To display output to the console device.

(3) Description of Parameters: A value of type Byte to be displayed is the input parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Outconsole implements the CP/M-86 function call #Ø2.

l. Send_block

(1) Type: Procedure

(2) Purpose: To send a block of data to another terminal via the modem port.

(3) Description of Parameters: A value of type Integer indicating the address of the data structure to be sent is one input parameter. A value of the type Integer indicating the size of the data structure is the other input parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Send_block sends a block of data out the modem port (J2) sequentially one byte at a time, decrementing the size of the data block until it equals zero. Then it transmits a sequence of four end of block codes (ØFFh) to indicate that end of block has been reached. Any data structure may be sent using this procedure.

m. Yes

(1) Type: Procedure

(2) Purpose: To indicate to the receiving terminal(s) that an end of process has been reached.

(3) Description of Parameters: None.

(4) Subroutines Called: N/A.

(5) Process Description:

Yes sends out a sequence of four end of process codes (ØF1h) when a particular process is finished.

n. No

(1) Type: Procedure

(2) Purpose: To indicate to the receiving terminal(s) that a process is still ongoing.

(3) Description of Parameters: None.

(4) Subroutines Called: N/A.

(5) Process Description:

No sends out a single no code (6Eh) to indicate that a process is not finished.

o. Recv_block

(1) Type: Procedure

(2) Purpose: To receive a block of data from another machine via the modem port.

(3) Description of Parameters: A value of type Integer indicating the address of the data structure in which the received data will be stored is the input parameter. A value of the type Integer indicating the amount of data received is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Recv_block receives a block of data from the modem port (J2) sequentially one byte at a time, incrementing the size of the data block until a sequence of four end of block codes (ØFFh) is received indicating that end of block has been reached. Any data structure may be received using this procedure.

p. Endfile

(1) Type: Procedure

(2) Purpose: To determine whether or not a file or directory transfer process has finished

(3) Description of Parameters: A value of the type Boolean indicating finish or not is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Endfile waits to receive either a single no code (6Eh) or a sequence of four end of process codes (ØF1h). If a no code is received the value false is returned to the output parameter, if a sequence of four end of process codes is received a value of true is returned.

q. Active

(1) Type: Procedure

(2) Purpose: To indicate to the Concentrator that a particular terminal is in the network.

(3) Description of Parameters: None.

(4) Subroutines Called: N/A.

(5) Process Description:

Active sends out an active code (ØDØh) when the terminal is ready to communicate with the network.

r. Waiting

(1) Type: Procedure

(2) Purpose: To receive the status the Concentrator has for an active terminal.

(3) Description of Parameters: A value of type byte indicating the status.

(4) Subroutines Called: N/A.

(5) Process Description:

Waiting receives the status from the Concentrator that a terminal's active code prompted. A Ø indicates nothing waiting, a 1 indicates file waiting, a 2 indicates message waiting, and a 3 indicates that the inactive receiving terminal to which this terminal had previously tried to send is now active.

s. Setup

(1) Type: Procedure

(2) Purpose: To establish a connection with another terminal via the Concentrator.

(3) Description of Parameters: A value of type Integer indicating the address of the connection data structure. A value of the type Boolean indicating the result of the connection set up attempt is the output variable.

(4) Subroutines Called: N/A.

(5) Process Description:

Setup transmits to the Concentrator the connection data structure for the purposes of establishing a connection with another terminal. The connection data structure consists of destination field for the destination terminal(s), a source field for the source terminal, and a process field for the process to be accomplished. Setup sends out a sequence of four end of block codes (ØFFh) to indicate the end of the data structure. It then waits to determine the result of the connection set up attempt. If the connection was successful, send_ready is set to true. If it was not because the destination was inactive, then send_ready is set to false.

t. Myput

(1) Type: Procedure

(2) Purpose: To display integers.

(3) Description of Parameters: A value of type Integer indicating the integer to be displayed.

(4) Subroutines Called: N/A.

(5) Process Description:

Myput converts the integer input into its ascii equivalent and uses CP/M-86 function call #Ø2 to display it.

u. Put_str

(1) Type: Procedure

(2) Purpose: To display strings.

(3) Description of Parameters: A value of type String indicating the address of the string to be displayed.

(4) Subroutines Called: N/A.
(5) Process Description:

Put_str is passed the address of the string to be displayed. The first byte at that address is the string length. The succeeding byte are displayed using CP/M-86 function call #Ø2 until the string length counter equals zero.

v. Endmsg
(1) Type: Procedure
(2) Purpose: To determine if end of message has occurred.
(3) Description of Parameters: None.
(4) Subroutines Called: N/A.
(5) Process Description:

Endmsg waits until a sequence of four end of message codes (ØF1h) is received.

w. Search_first
(1) Type: Procedure
(2) Purpose: To search for the first directory match.
(3) Description of Parameters: A value of the type Integer indicting the buffer address for the directory record is one input parameter. A value of the type Integer indicting the FCB address is the other input parameter. A value of the type Integer indicting the result of the function is one output parameter. A value of the type Integer indicting the location of the matched directory entry is the other output parameter.
(4) Subroutines Called: N/A.
(5) Process Description:

Search_first implements CP/M-86 function call #17. It searches the directory of the specified drive for the first match of the file name and type of the specified FCB. When a successful match is found, Search_first places the 128 byte record containing the matched directory entry and returns an offset code (Ø, 1, 2, or 3) that specifies the exact location of the entry within the record using the formula: location of entry = (offset * 32) + DMA. If a match cannot be found, then Search_first returns 255 (ØFFh).

x. Search_next
(1) Type: Procedure
(2) Purpose: To search for the next directory match.
(3) Description of Parameters: A value of the type Integer indicting the buffer address for the directory record is one input parameter. A value of the type Integer indicting the FCB address is the other input parameter. A value of the type Integer indicting the result of the function is one output parameter. A value of the type Integer

indicting the location of the matched directory entry is the other output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Search_next implements CP/M-86 function call #18. It searches the directory of the specified drive for the next match of the file name and type of the specified FCB. When a successful match is found, Search_first places the 128 byte record containing the matched directory entry and returns an offset code (Ø, 1, 2, or 3) that specifies the exact location of the entry within the record using the formula: location of entry = (offset * 32) + DMA. If a match cannot be found, then Search_first returns 255 (ØFFh).

y. End_block

(1) Type: Procedure

(2) Purpose: To indicate end of block during directory transfer.

(3) Description of Parameters: None.

(4) Subroutines Called: N/A.

(5) Process Description:

End_block transmits a sequence of four end of block codes (ØFFh) to indicate end of block during directory transfers.

z. Send_string

(1) Type: Procedure

(2) Purpose: To transmit strings.

(3) Description of Parameters: A value of type String indicating the address of the string to be output is the input parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Send_string is passed the address of the string to be transmitted. The first byte of the input parameter is the length of the string. Send_string transmits the succeeding bytes one at a time out the modem port (J2) until the string length counter equals zero.

aa. Send_dir

(1) Type: Procedure

(2) Purpose: To transmit matched directory entries.

(3) Description of Parameters: A value of type Integer indicating the address of the matched directory to be output is one input parameter. A value of the type Integer indicating the size of the directory entry to be transmitted is the other input parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Send_dir transmits the bytes of the matched directory entry sequentially, one byte at a time, via the modem port (J2) until the directory length counter equals zero.

bb. Driveout
(1) Type: Procedure
(2) Purpose: To transmit the disk drive used in a directory transfer.
(3) Description of Parameters: A value of type Byte indicating the specified drive to be output is the input parameter.
(4) Subroutines Called: N/A.
(5) Process Description:
Driveout transmits the byte representing the specified drive via the modem port (J2).

cc. Drivein
(1) Type: Procedure
(2) Purpose: To receive the disk drive used in a directory transfer.
(3) Description of Parameters: A value of type Byte indicating the specified drive received is the output parameter.
(4) Subroutines Called: N/A.
(5) Process Description:
Drivein receives the byte representing the specified drive via the modem port (J2).

dd. Off
(1) Type: Procedure
(2) Purpose: To inform the Concentrator that a terminal is no longer active.
(3) Description of Parameters: None.
(4) Subroutines Called: N/A.
(5) Process Description:
Off transmits the off code (ØFh) via the modem port (J2).

J. NAMES
1. CONFIGURATION
a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Intel 86/12A SBC
e. Operating System - CP/M-86 (version 1.14)
f. Package description:
The Names specification contains the following global objects:
TYPE conctn
connection
TYPE blk
block
block_size
retrn

```
space
input
pause
confirm
drive
code
dest
srce
send_ready
```

APPENDIX C

MAINTENANCE MANUAL FOR CONCENTRATOR PROGRAMS

A. POLL

1. CONFIGURATION

a. Language - JANUS/Ada

b. Compiler Version - 1.47

c. Linker Version - 1.47

d. Target Hardware - Intel 86/12A SBC

e. Operating System - CP/M-86 (version 1.14)

f. Package description:

The Poll package is the main program for the Concentrator acting as a network switchboard. It contains an infinite loop that polls each one of 23 ports (bypassing port #24 which is designated as the bulletin board port and is never polled) continuously. Poll controls the satisfying of requests from the Z-100 workstations and the storing of those requests in a FIFO queue that cannot be satisfied. For each port Poll checks the queue for waiting processes. If there is a process waiting, Poll will satisfy the waiting process and then attempt to satisfy the polled port's original request. If not, Poll will immediately attempt to satisfy the polled port's request. Poll is responsible for decoding each workstation's request and establishing the proper path between sender and receiver(s). It is also responsible for maintaining the net status; the list of all currently active ports, and transmitting it to workstations requesting it.

2. SUBROUTINES

a. Convert

(1) Type: Procedure

(2) Purpose: To convert the byte information received from the Z-100 workstations concerning source and destination terminals into their integer physical addresses.

(3) Description of Parameters: A value of type integer indicating the index position in the queue of the current process_status record is the input variable.

(4) Subroutines Called: None

(5) Process Description:

Convert is two large case statements which translate the bytes received from the workstations into the proper integer physical addresses for each port.

b. Poll calls the following subroutines:

Check_port

Check_queue

Connect

Convert

Xfer

Concxfer

No_xfer
Broadcast
Net_stat
Queue_status

3. COMMENTS

Poll is resident in the Concentrator and is invoked by typing the command "poll". It will not function properly if all three 8538 BLC expansion boards are not installed.

To change the port number of the bulletin board, the polling sequence must be changed to bypass the new bulletin board port and the automatic routing routines that route bulletin board requests to the predesignated port must be changed accordingly. Additionally the value of 'bullport' in the procedure Broadcast must be changed.

To expand the system for greater numbers of terminals, add the appropriate number of expansion boards and change the following constants: 'boardno', 'machno', 'maxque', and the procedure Convert which contains the physical addresses. Additionally, the value of 'boardnum' must be changed in the procedure Broadcast.

B. CONCUTIL

1. CONFIGURATION

a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Intel 86/12A SBC
e. Operating System - CP/M-86 (version 1.14)
f. Package description:

The Concutil package contains utility programs for the Concentrator.

2. SUBROUTINES

a. Check_queue

(1) Type: Procedure

(2) Purpose: To check the queue for waiting processes.

(3) Description of Parameters: A value of type Integer indicating the number of processes waiting in the queue is one input parameter. A value of type Integer indicating the port for whom waiting processes are being checked is the other input para-meter. A value of type Integer indicating the position index of the waiting process is one output variable. A value of type Boolean indicating whether or not there are waiting processes is the other output variable.

(4) Subroutines Called:
Check_port
Queue_status

(5) Process Description:

Check_queue checks the queue for waiting processes. If there are none in the queue then false is returned. If there are processes in the queue, a check is made if they are waiting for the specified port. If not, false is returned. If there are processes in the queue and they are waiting for the specified port, their position in the queue is returned along with a value of true. Check_queue then tells the specified port what is waiting for it and tells the sender of the waiting process to resend.

b. Net_stat

(1) Type: Procedure

(2) Purpose: To decode and transmit the list of active terminals in the network to the requestor.

(3) Description of Parameters: A value of type integer indicating the port for which the net status is intended.

(4) Subroutines Called:

Send_who_block

(5) Process Description:

Net_stat controls the transmission of the list of active network terminals. Net_stat checks the data structure 'active_list' for those terminals for which true has been recorded, indicating that they are active and stores the terminal number information in the data structure 'who_list'. It is 'who_list' that is transmitted to the requestor.

C. COASMLIB

1. CONFIGURATION

a. Language - JANUS/Assembler

b. Compiler Version - 1.50

c. Linker Version - 1.50

d. Target Hardware - Intel 86/12A SBC

e. Operating System - CP/M-86 (version 1.14)

f. Package description:

The Coasmlib package contains the library of assembly language subroutines for the Concentrator.

2. COMMENTS

a. JANUS/Ada parameters for JANUS/Assembler modules are placed on the stack at subroutine call with the last parameter closest to the top and the return address on the very top. Discrete values are passed for parameters of type IN and the address of the parameter is passed for types OUT and IN OUT. Upon return to the calling program OUT and IN OUT parameters are removed from the stack along with the return address.

b. The following subroutines involve input and output:

Check_port
Connect
Queue_status
Send_who_block
Broadcast
Concxfer
Xfer
No_xfer

The method of transmission used involves an immediate echo checking procedure. The sending subroutine transmits bytes of data one at a time. The receiving subroutine receives the data bytes and echos each one immediately upon reception. The sending subroutine compares the echo with the transmitted data and checks for error. If an error is detected, the sending subroutine either simply retransmits or sends an error code to tell the receiving subroutine that the data previously received (and processed) was in error, followed by the retransmitted data. All receiving subroutines contain a finite waiting time after the last data was received to ensure that the final echo was received properly.

3. SUBROUTINES

a. Check_port

(1) Type: Procedure

(2) Purpose: To poll network ports looking for the active signal

(3) Description of Parameters: A value of type Integer containing the address of the port to be polled is the input variable. A value of type Boolean indicating the result of the poll is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Check_port polls the port indicated looking for the active signal (ØDØh). It will poll for a finite period of time. If no signal, or the wrong signal, is found, ready is set to false. If the active signal is found, ready is set to true. Check_port is used for two purposes. It is used to initially poll ports for incoming requests (indicated by active signal) and to poll a destination port to determine if it is ready to receive.

b. Connect

(1) Type: Procedure

(2) Purpose: To receive the connection record transmitted by the sending Z-100.

(3) Description of Parameters: A value of type Integer containing the address of the port from which the connection record is to be received is the input parameter. A value of type Integer indicating the address of the data structure in which the connection record is to be stored is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Connect receives the connection record, sequentially, one byte at a time and stores it in the data structure indicated by the input parameter until a sequence of four finish codes (ØFFh), indicating end of block, are received.

c. Queue_status

(1) Type: Procedure

(2) Purpose: To inform a polled port of the status of waiting processes in the queue.

(3) Description of Parameters: A value of type Integer containing the address of the polled port is one input parameter. A value of type Byte indicating the process waiting in the queue (or none) is the other output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Queue_status transmits to the port indicated by one input parameter the byte value of the process waiting for that port (or none).

d. Send_who_block

(1) Type: Procedure

(2) Purpose: To transmit the data structure containing the net status.

(3) Description of Parameters: A value of type Integer containing the address of the port to which the net status is destined is one input parameter. A value of type Integer indicating the address of the data structure containing the net status is another input parameter. A value of type Integer indicating the length of the data structure is the last output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Send_who_block transmits the data structure containing the net status sequentially, one byte at a time, until the length counter equals zero. Then it sends out a sequence of four finish codes (ØFFh), indicating end of block.

e. Broadcast

(1) Type: Procedure

(2) Purpose: To transfer data in a broadcast mode: from one to many.

(3) Description of Parameters: A value of type Integer containing the address of the sending port is the input parameter. A value of type Byte containing the address used to store the input temporarily is the output parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Broadcast controls the transmission of data from the sending port to all receiving ports. The data is transmitted one byte at a time as rapidly as possible to all ports (bypassing the sender, and the bulletin board). The first byte is received from the sending port specified in the first input parameter. It is then sent in quick succession to all the ports beginning with the first port in the network. Broadcast increments the receiving port until all the ports are addressed and the byte is transmitted. The input is saved temporarily for error checking. Then Broadcast loops back to receive the echoes. If there are any echoes in error, that port address is saved until all the echoes have been received. Then all the addresses in error are serviced, one at a time until the error is corrected. At this point, or if there were no errors, Broadcast echoes back to the sender the received byte for the sender's error checking. If there was an error, the sender sends out an error code then the retransmitted byte and the entire process is repeated until the error is corrected. If there is no error, the next byte is transmitted to the Concentrator until a sequence of four end of process code echoes (ØF1h) are received from all the receiving terminals. Broadcast waits only a finite period of time for echoes from each receiving terminal, so if a terminal is inactive, the bytes transmitted to it will be continuously overwritten. No attempt is made to bypass inactive terminals.

f. Concxfer

(1) Type: Procedure

(2) Purpose: To transfer data from a sending terminal to a single destination terminal.

(3) Description of Parameters: A value of type Integer containing the address of the sending port is one input parameter. A value of type Integer containing the address of the receiving port is the other input parameter.

(4) Subroutines Called: N/A.

(5) Process Description:

Concxfer controls the transmission of data between two terminals. The first byte is received from the sending port indicated by the first input parameter and transmitted from the receiving port indicated by the second parameter. The echo is then received from the receiver and transmitted to the sender. Concxfer performs no error checking; it merely passes data and echoes back and forth until it receives a sequence of four end of process code echoes (ØF1h) from the receiver. At that point it terminates the connection.

g. Xfer
(1) Type: Procedure
(2) Purpose: To inform the sending port that a connection has been established.
(3) Description of Parameters: A value of type Integer containing the address of the sending port is the input parameter.
(4) Subroutines Called: N/A.
(5) Process Description:
Xfer transmits to the sending port a code (Ø1h) informing it that a connection has been established.

h. No_xfer
(1) Type: Procedure
(2) Purpose: To inform the sending port that a connection can not be established.
(3) Description of Parameters: A value of type Integer containing the address of the sending port is the input parameter.
(4) Subroutines Called: N/A.
(5) Process Description:
No_xfer transmits to the sending port a code (ØØh) informing it that a connection can not be established.

D. CONCNAME
1. CONFIGURATION
a. Language - JANUS/Ada
b. Compiler Version - 1.47
c. Linker Version - 1.47
d. Target Hardware - Intel 86/12A SBC
e. Operating System - CP/M-86 (version 1.14)
f. Package description:
The Concname specification contains the following global objects:
TYPE process_status
max_que
TYPE que
queue
resend
zero
machno
ready
active_list

APPENDIX D

LISTING OF Z-100 PROGRAMS

```
PACKAGE Names IS

--* GLOBAL TYPES, CONSTANTS, AND VARIABLES *--

   TYPE conctn IS
      RECORD
         process: byte;
         source: byte;
         destination: byte;
      END RECORD;

   connection: conctn;

   TYPE blk IS ARRAY (1..132) OF byte;

   block: blk;

   block_size: CONSTANT Integer := 128;
   retrn: CONSTANT BYTE := byte (16#0D#);
   space: CONSTANT BYTE := byte (16#20#);
   input: byte;
   pause: byte;
   confirm: byte;
   drive: byte;
   code: integer;
   dest, srce: Integer;
   send_ready: Boolean;

END Names;

================================================================

PACKAGE Xferfile IS

   TYPE fcb IS
      RECORD
         dr: byte;
         fn: ARRAY (1..8) OF byte;
         ft: ARRAY (1..3) OF byte;
         ex: byte;
         s1: byte;
         s2: byte;
         rc: byte;
         dn: ARRAY (1..16) OF byte;
         cr: byte;
      END RECORD;
```

```
   fcb1: fcb;
   fn_length: Integer;

   PROCEDURE Sendfile;
   PROCEDURE Receivefile;
   PROCEDURE Parse_cap (fcb2: OUT fcb; fnlen: OUT Integer);
END Xferfile;



WITH Myutil, Myasmlib, Names;
PACKAGE BODY Xferfile IS
   USE Myutil, Myasmlib, Names;


   PROCEDURE Sendfile IS

   --* AUTHOR: THOMAS V. WORKS
   --* DATE: JULY 1986
   --* DESCRIPTION:   SENDFILE PROMPTS THE USER FOR FILE NAME *--
   --* AND TYPE,  OPENS THE FILE AND TRANSMITS EACH 128 BYTE  *--
   --* RECORD UNTIL END OF FILE.   UPON COMPLETION,  SENDFILE *--
   --* CLOSES THE FILE AND RETURNS TO MAIN MENU               *--

   xbytes: integer := 0;
   ctrl_f: CONSTANT BYTE := byte (16#06#);

   BEGIN
      Clearscrn;

      --* SET FCB FOR FILE OPERATIONS *--

      fcb1.dr := byte(0);
      fcb1.ex := byte(0);
      Reset_disk;

      New_line;

      --* PROMPT USER FOR DISK DRIVE *--

      Drive_select (drive);
      New_line;

      --* PROMPT USER FOR FILE NAME AND TYPE *--

      LOOP
         Put ("ENTER FILE NAME."); New_line;
         LOOP
            Put ("FN.FT: ");
            Parse_cap (fcb1, fn_length);
            Put ("FILE ");
            Outconsole (drive);
```

```
      Put (":");
      FOR i IN 1..fn_length LOOP
         Outconsole (fcb1.fn (i) );
      END LOOP;
      Put (".");
      FOR i IN 1..3 LOOP
         Outconsole (fcb1.ft (i) );
      END LOOP;
      Put (" IS SELECTED. PRESS RETURN TO CONFIRM, ");
      Put ("ANY OTHER KEY TO RESELECT.");
      Keyin (confirm);
      New_line;
      IF confirm = retrn THEN
         EXIT;
      END IF;
      Put ("ENTRY CANCELLED. REENTER FN:FT."); New_line;
      New_line;
   END LOOP;

   Open_file (fcb1'ADDRESS, code);
   fcb1.cr := byte (0);

   --* SET DMA TO ADDRESS OF  DATA STRUCTURE THAT WILL  *--
   --* HOLD DATA READ FROM FILE                         *--

   Set_DMA (block'ADDRESS);

   --* IF CODE = 255 (0FFh) THEN FILE COULD NOT BE      *--
   --* FOUND AND PROMPT USER TO REENTER FILE NAME AND   *--
   --* TYPE                                             *--

   IF code /= 255 THEN
      Put ("FILE ");
      Outconsole (drive);
      Put (":");
      FOR i IN 1..fn_length LOOP
         Outconsole (fcb1.fn (i) );
      END LOOP;
      Put (".");
      FOR i IN 1..3 LOOP
         Outconsole (fcb1.ft (i) );
      END LOOP;
      Put (" IS OPENED."); New_line;
      Clearscrn;
      EXIT;
   END IF;
   Put ("FILE NOT FOUND. PLEASE TRY AGAIN."); New_line;
END LOOP;

--* PROMPT USER FOR DESTINATION AND SOURCE TERMINAL #'s *--

Enter_machine (dest, srce);
```

```
    --* CREATE CONNECTION RECORD *--

    connection.destination := byte (dest);
    connection.source := byte (srce);
    connection.process := ctrl_f;

    --* ESTABLISH CONNECTION WITH DESTINATION *--

    Put ("WAITING..."); New_line; New_line;
    Setup (connection'ADDRESS, send_ready);
    IF send_ready THEN
       IF dest = Ø THEN
          Put ("FOR BROADCAST, PAUSE TO ALLOW RECEIVER TO ");
          Put ("GET READY"); New_line;
          Put ("PRESS ANY KEY TO SEND. ");
          Keyin (pause); New_line; New_line;
       END IF;
       Put ("CONNECTION ESTABLISHED, SENDING FILE... ");
       New_line; New_line;

       --* READ AND SEND FILE IN 128 BYTE BLOCKS *--

       LOOP
          Read_seq (fcb1'ADDRESS, code);
          IF code = 1 THEN
             Yes;
             EXIT;
          ELSE
             No;
          END IF;
          Send_block (block'ADDRESS, block_size);
       END LOOP;

       Close_file (fcb1'ADDRESS, code);
       Put ("FILE SENT."); New_line;
    ELSE
       Put ("FILE NOT SENT. DESTINATION INACTIVE."); New_line;
    END IF;
    Put ("PRESS ANY KEY TO CONTINUE. ");
    Keyin (pause);

END Sendfile;

-----------------------------------------------------------

PROCEDURE Receivefile IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: JULY 1986
--* DESCRIPTION:  RECEIVEFILE PROMPTS THE USER FOR FILE    *--
--* NAME AND TYPE, CREATES THE FILE, AND RECEIVES THE 128  *--
```

```
--* BYTE BLOCK FROM THE SOURCE AND WRITES TO DISK UNTIL     *-
--* END OF FILE. THEN IT CLOSES THE FILE AND DISPLAYS # OF *-
--* BYTES RECEIVED.  LASTLY IT RETURNS TO MAIN MENU.       *-

rbytes: Integer := 0;
length: Integer;
reason2: STRING := "END OF FILE.";
finished: Boolean := false;

BEGIN
   Clearscrn;

   --* SET FCB FOR FILE OPERATIONS *--

   fcb1.dr := byte(0);
   fcb1.ex := byte(0);
   Reset_disk;

   --* PROMPT USER FOR DISK DRIVE *--

   New_line;
   Drive_select (drive);
   New_line;
   Put ("ENTER FILE NAME."); New_line;

   --* PROMPT USER FOR FILE NAME AND TYPE *--

   LOOP
      Put ("FN.FT: ");
      Parse_cap (fcb1, fn_length);
      New_line;
      Put ("FILE ");
      Outconsole (drive);
      Put (":");
      FOR i IN 1..fn_length LOOP
         Outconsole (fcb1.fn (i) );
      END LOOP;
      Put (".");
      FOR i IN 1..3 LOOP
         Outconsole (fcb1.ft (i) );
      END LOOP;
      Put (" IS SELECTED. PRESS RETURN TO CONFIRM, ");
      New_line;
      Put ("ANY OTHER KEY TO RESELECT.");
      Keyin (confirm);
      New_line;
      IF confirm = retrn THEN
         EXIT;
      END IF;
      Put ("ENTRY CANCELLED. REENTER FN:FT."); New_line;
      New_line;
   END LOOP;
```

```
    Delete_file (fcb1'ADDRESS, code);
    Create_file (fcb1'ADDRESS, code);
    fcb1.cr := byte (0);

       --* SET DMA TO ADDRESS OF  DATA STRUCTURE THAT WILL  *--
       --* HOLD DATA RECEIVED FROM SOURCE                   *--

    Set_DMA (block'ADDRESS);
    IF code = 255 THEN
       Put ("DIRECTORY SPACE UNAVAILABLE.");
    ELSE
       Put ("FILE ");
       Outconsole (drive);
       Put (":");
       FOR i IN 1..fn_length LOOP
          Outconsole (fcb1.fn (i) );
       END LOOP;
       Put (".");
       FOR i IN 1..3 LOOP
          Outconsole (fcb1.ft (i) );
       END LOOP;
       Put (" IS OPENED."); New_line;
       Clearscrn;

       --* RECEIVE AND WRITE FILE IN 128 BYTE BLOCKS *--

       Put ("RECEIVING FILE..."); New_line;
       LOOP
          Endfile (finished);
          EXIT WHEN finished;
          Recv_block (block'ADDRESS, length);
          Write_seq (fcb1'ADDRESS, code);
       END LOOP;

       IF code = 1  THEN
          Put ("ERROR. NO AVAILABLE DIRECTORY SPACE.");
          New_line;
       ELSIF code = 2 THEN
          Put ("ERROR. DISK FULL."); New_line;
       ELSE
          Put ("FINISHED WRITING FILE."); New_line; New_line;
       END IF;

       Close_file (fcb1'ADDRESS, code);
       rbytes := rbytes * 128;
       Put_int (rbytes); Put (" BYTES RECEIVED."); New_line;
    END IF;
    Put ("PRESS ANY KEY TO CONTINUE. ");
    Keyin (pause);

END Receivefile;
```

```
PROCEDURE Parse_cap (fcb2: OUT fcb; fnlen: OUT Integer) IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  PARSE_CAP PARSES THE USER'S FILE NAME     *--
--* AND TYPE FOR INVALID CP/M-86 CHARACTERS, CHANGES INPUT *--
--* TO UPPERCASE, AND PLACES IT IN APPROPRIATE FIELDS OF   *--
--* FCB.                                                    *--

period: CONSTANT BYTE := byte (16#2E#);
lthan: CONSTANT BYTE := byte(16#3C#);
gthan: CONSTANT BYTE := byte(16#3E#);
comma: CONSTANT BYTE := byte(16#2C#);
semic: CONSTANT BYTE := byte(16#3B#);
colon: CONSTANT BYTE := byte(16#3A#);
equal: CONSTANT BYTE := byte(16#3D#);
qmark: CONSTANT BYTE := byte(16#3F#);
star:  CONSTANT BYTE := byte(16#2A#);
lbrac: CONSTANT BYTE := byte(16#5B#);
rbrac: CONSTANT BYTE := byte(16#5D#);

TYPE name IS ARRAY (1..8) OF byte;
filename: name;
capital: Integer;
k: Integer := 0;
h: Integer := 0;


BEGIN

   --* PARSE FILE NAME *--

   LOOP
      Keyin (input);
      Outconsole (input);
      CASE input IS
         WHEN lthan!gthan!comma!semic!colon!
              equal!qmark!star!lbrac!rbrac
              => New_line; Put ("INVALID. TRY AGAIN.");
                 New_line;
                 FOR i IN 1..k LOOP
                    Outconsole (filename (i));
                 END LOOP;

         WHEN period => EXIT;
         WHEN OTHERS
              => k := k + 1;
                 filename (k) := input;

         END CASE;
      EXIT WHEN k = 8;  --* MAX FILE NAME LENGTH *--
   END LOOP;
```

```
IF k = 8 THEN
   Put (".");
END IF;

--* PARSE FILE TYPE *--

LOOP
   Keyin (input);
   Outconsole (input);
   CASE input IS
      WHEN lthan!gthan!comma!semic!colon!
           equal!qmark!star!lbrac!rbrac!period
           => New_line; Put ("INVALID. TRY AGAIN.");
              New_line;
              FOR I IN 1..k LOOP
                 Outconsole (filename (i));
              END LOOP;
              Put (".");
              FOR j IN 1..h LOOP
                 Outconsole (fcb2.ft (j));
              END LOOP;

      WHEN OTHERS
           => h := h + 1;
              fcb2.ft (h) := input;

   END CASE;
   EXIT WHEN h = 3;  --* MAX FILE TYPE LENGTH *--
END LOOP;

--* PLACE PARSED INPUT IN FCB ADDING BLANKS TO FILL UP  *--
--* FIELDS                                              *--

fnlen := k;
FOR i IN 1..8 LOOP
   fcb2.fn (i) := filename (i);
   IF i = fnlen THEN
      FOR j IN (fnlen + 1)..8 LOOP
         fcb2.fn (j) := space;
      END LOOP;
      EXIT;
   END IF;
END LOOP;
New_line;

--* CAPITALIZE FILE NAME AND TYPE *--

FOR i IN 1..fnlen LOOP
   capital := Integer (fcb2.fn (i));
   CASE capital IS
      WHEN 16#61#..16#7A# => capital := capital - 16#20#;
```

```
                                    => NULL;
          WHEN OTHERS
       END CASE;
       fcb2.fn (i) := byte (capital);
    END LOOP;

    FOR j IN 1..3 LOOP
       capital := Integer (fcb2.ft (j));
       CASE capital IS
          WHEN 16#61#..16#7A# => capital := capital - 16#20#;
          WHEN OTHERS        => NULL;
       END CASE;
       fcb2.ft (j) := byte (capital);
    END LOOP;

  END Parse_cap;

END Xferfile;

================================================================

PACKAGE Messages IS
   PROCEDURE Talking;
   PROCEDURE Listening;
END Messages;


WITH Myasmlib, Myutil, Names;
PACKAGE BODY Messages IS
   USE Myasmlib, Myutil, Names;

    end_of_msg: CONSTANT BYTE := byte (16#1A#); --control Z--
    max_msg_length: CONSTANT := 1600;
    max_line_length: CONSTANT := 80;
    message: ARRAY (1..max_msg_length) OF byte;
    msg_length: Integer;
    line_length: Integer;

    PROCEDURE Talking IS

    --* AUTHOR: THOMAS V. WORKS
    --* DATE: AUGUST 1986
    --* DESCRIPTION:  TALKING PROMPTS USER TO TYPE MESSAGE,     *--
    --* STORES MESSAGE IN DATA STRUCTURE AND TRANSMITS MESSAGE *--
    --* TO DESTINATION AS A SINGLE BLOCK.                      *--

        ctrl_m: CONSTANT BYTE := byte (16#0D#);
        ctrl_s: CONSTANT BYTE := byte (16#13#);
        response: byte;
```

```
BEGIN
   LOOP
      Clearscrn;
      Put ("BEGIN TYPING MESSAGE. USE CTRL Z TO STOP.");
      New_line;
      Put ("MAXIMUM MESSAGE LENGTH IS 1600 CHARACTERS.");
      New_line; New_line;
      line_length := 0;

      --* TYPE IN PAGE FORMAT; 20 LINES, 80 CHARACTERS  *--
      --* PER LINE                                      *--

      FOR i IN 1..max_msg_length LOOP
         Keyin (input);
         EXIT WHEN input = end_of_msg;
         message (i) := input;
         msg_length := i;
         line_length := line_length + 1;
         Outconsole (input);
         IF (input = retrn) OR
            (line_length = max_line_length) THEN
            New_line;
            line_length := 0;
         END IF;
      END LOOP;
      New_line;

      Put ("END OF MESSAGE."); New_line; New_line;
      Put ("TYPE CTRL_S TO SEND. ANY OTHER KEY TO RETYPE ");
      Put ("MESSAGE."); New_line;
      Keyin (response);
      IF response = ctrl_s THEN
         EXIT;
      END IF;
      Put ("MESSAGE ERASED."); New_line;
   END LOOP;
   New_line;
   Put ("IF YOU WANT TO SEND TO THE BULLETIN BOARD,");
   New_line;
   Put ("ENTER 24 FOR DESTINATION MACHINE.");
   New_line; New_line;

--* PROMPT USER FOR DESTINATION AND SOURCE TERMINAL #'s *--

   Enter_machine (dest, srce);

--* CREATE CONNECTION RECORD *--

   connection.destination := byte (dest);
   connection.source := byte (srce);
   connection.process := ctrl_m;
```

```
   --* ESTABLISH CONNECTION WITH DESTINATION *--

      Put ("WAITING..."); New_line; New_line;
      Setup (connection'ADDRESS, send_ready);
      IF send_ready THEN
         IF dest = 0 THEN
            Put ("FOR BROADCAST, PAUSE TO ALLOW RECEIVER TO "
            Put ("GET READY"); New_line;
            Put ("PRESS ANY KEY TO SEND. ");
            Keyin (pause); New_line; New_line;
         END IF;
         Put ("CONNECTION ESTABLISHED, SENDING MESSAGE... ");
         New_line; New_line;

         --* TRANSMIT MESSAGE AS A SINGLE BLOCK *--

         Send_block (message'ADDRESS, msg_length);
         Yes;

         Put ("MESSAGE SENT."); New_line;
      ELSE
         Put ("MESSAGE NOT SENT. DESTINATION INACTIVE.");
         New_line;
      END IF;
      Put ("PRESS ANY KEY TO CONTINUE. ");
      Keyin (pause);

END Talking;

PROCEDURE Listening IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  LISTENING RECEIVES TRANSMITTED MESSAGE   *--
--* AND DISPLAYS IT IN A PAGE FORMAT OF 20 LINES, 80       *--
--* CHARACTERS PER LINE.                                   *--

char: byte;

   BEGIN
      Clearscrn;
      Put ("RECEIVING MESSAGE..."); New_line;

      --* RECEIVE ENTIRE MESSAGE *--

      Recv_block (message'ADDRESS, msg_length);
      Endmsg;
      Put_int (msg_length); Put (" BYTES RECEIVED."); New_line

         --* DISPLAY IN SAME PAGE FORMAT AS TYPED *--

      line_length := 0;
```

```
         FOR i IN Ø..(msg_length - 1) LOOP
            char := message (i);
            line_length := line_length + 1;
            Outconsole (char);
            IF (char = return) OR
               (line_length = max_line_length) THEN
               New_line;
               line_length := Ø;
            END IF;
         END LOOP;

         New_line; New_line;
         Put ("END OF MESSAGE."); New_line;
         Put ("PRESS ANY KEY TO CONTINUE. ");
         Keyin (pause);

      END Listening;

END Messages;

================================================================

PACKAGE Directry IS
   PROCEDURE Present_dir;
   PROCEDURE Receive_dir;
END Directry;


WITH Xferfile, Myutil, Myasmlib, Names;
PACKAGE BODY Directry IS
   USE Xferfile, Myutil, Myasmlib, Names;


   PROCEDURE Present_dir IS

   --* AUTHOR: THOMAS V. WORKS                                    *--
   --* DATE: AUGUST 1986                                          *--
   --* DESCRIPTION:  PRESENT_DIR PROMPTS THE USER FOR THE         *--
   --* REQUESTED DISK DRIVE AND TRANSMITS THE DIRECTORY EIGHT     *--
   --* ENTRIES AT A TIME (128 BYTES) UNTIL THE ENTIRE             *--
   --* DIRECTORY IS SENT.                                         *--

      nodir_str: STRING := "NO DIRECTORY ON SELECTED DRIVE.";
      dir_size: CONSTANT := 16;
      count: Integer;
      ctrl_d: CONSTANT BYTE := byte (16#Ø4#);
      qmark: CONSTANT BYTE := byte (16#3F#);

      TYPE buff IS ARRAY (1..128) OF byte;
      buffer: buff;
      dir_addr: Integer;
```

```
BEGIN
   Clearscrn;
   fcb1.dr := byte(0);
   fcb1.ex := byte(0);
   Reset_disk;

   --* WILD CARD FOR MATCHING ENTIRE DIRECTORY *--

   FOR i IN 1..8 LOOP
      fcb1.fn (i) := qmark;
   END LOOP;
   FOR i IN 1..3 LOOP
      fcb1.ft (i) := qmark;
   END LOOP;

   New_line;

   --* PROMPT USER FOR DISK DRIVE *--

   Drive_select (drive);
   New_line;

   --* PROMPT USER FOR DESTINATION AND SOURCE TERMINAL #'s

   Enter_machine (dest, srce);

   --* CREATE CONNECTION RECORD *--

   connection.destination := byte (dest);
   connection.source := byte (srce);
   connection.process := ctrl_d;

   --* ESTABLISH CONNECTION WITH DESTINATION *--

   Put ("WAITING..."); New_line; New_line;
   Setup (connection'ADDRESS, send_ready);
   IF send_ready THEN
      IF dest = 0 THEN
         Put ("FOR BROADCAST, PAUSE TO ALLOW RECEIVER TO ")
         Put ("GET READY"); New_line;
         Put ("PRESS ANY KEY TO SEND. ");
         Keyin (pause); New_line; New_line;
      END IF;

      Put ("CONNECTION ESTABLISHED, SENDING DIRECTORY ");
      Put ("FROM DRIVE ");
      Outconsole (drive); Put (" ... "); New_line; New_line

      --* SET DMA TO ADDRESS OF  DATA STRUCTURE THAT     *--
      --* WILL HOLD DIRECTORY ENTRIES                    *--

      Set_DMA (buffer'ADDRESS);
```

```
--* TRANSMIT DRIVE *--

Driveout (drive);

--* MAKE FIRST DIRECTORY MATCH *--

Search_first (fcb1'ADDRESS, buffer'ADDRESS, code,
              dir_addr);
IF code = 255 THEN

--* NO MATCH, TRANSMIT NO DIRECTORY STRING *--

   Put_str (nodir_str); New_line;
   No;
   Send_string (nodir_str);
   End_block;
   Yes;
ELSE

   --* TRANSMIT DIRECTORY ENTRY *--

   No;
   Send_dir (dir_addr, dir_size);
   count := 1;
   LOOP
      LOOP

         --* MAKE SUCCESSIVE MATCHES UNTIL END    *--
         --* OF DIRECTORY (CODE = 255)            *--

         Search_next (fcb1'ADDRESS, buffer'ADDRESS,
                      code, dir_addr);
         EXIT WHEN code = 255;

         --* TRANSMIT 8 ENTRIES PER BLOCK *--

         Send_dir (dir_addr, dir_size);
         count := count + 1;
         IF count = 8 THEN
            count := 0;
            EXIT;
         END IF;
      END LOOP;
      IF code = 255 THEN
         No;
         End_block;
         EXIT;
      END IF;
      End_block;
      No;
```

```
                --* PAUSE AFTER EACH BLOCK FOR BROADCAST *--

                IF dest = 0 THEN
                   Put ("BROADCAST, PRESS ANY KEY TO SEND. ");
                   Keyin (pause); New_line;
                END IF;
             END LOOP;
             Yes;
          END IF;
          Put ("DIRECTORY SENT."); New_line;
       ELSE
          Put ("DIRECTORY NOT SENT. DESTINATION INACTIVE.");
          New_line;
       END IF;
       Put ("PRESS ANY KEY TO CONTINUE. ");
       Keyin (pause);

END Present_dir;

-------------------------------------------------------------------

PROCEDURE Receive_dir IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  RECEIVE_DIR PROMPTS THE USER FOR THE      *--
--* FILE NAME AND TYPE IN WHICH HE WISHES TO STORE ENTRIES *--
--* RECEIVED.  USER CAN SELECT WHICH BLOCK(S) OF 8 ENTRIES *--
--* HE WISHES TO SAVE, WHICH ARE THEN WRITTEN TO DISK.      *--

   choice: byte;
   rbytes: integer := 0;
   length: Integer;
   finished: Boolean := false;
   k: Integer;
   start, fini: Integer;
   period: CONSTANT BYTE := byte (16#2E#);
   no: CONSTANT BYTE := byte (16#6E#);

   BEGIN
      Clearscrn;

      --* SET FCB FOR FILE OPERATIONS *--

      fcb1.dr := byte(0);
      fcb1.ex := byte(0);
      Reset_disk;

      --* PROMPT USER FOR DISK DRIVE *--

      New_line;
      Drive_select (drive);
```

```
New_line;

--* PROMPT USER FOR FILE NAME AND TYPE *--

Put ("ENTER FILE NAME TO STORE YOUR DIRECTORY.");
New_line;
LOOP
   Put ("FN.FT: ");
   Parse_cap (fcb1, fn_length);
   New_line;
   Put ("FILE ");
   Outconsole (drive);
   Put (":");
   FOR i IN 1..fn_length LOOP
      Outconsole (fcb1.fn (i) );
   END LOOP;
   Put (".");
   FOR i IN 1..3 LOOP
      Outconsole (fcb1.ft (i) );
   END LOOP;
   Put (" IS SELECTED. PRESS RETURN TO CONFIRM, ");
   New_line;
   Put ("ANY OTHER KEY TO RESELECT.");
   Keyin (confirm);
   New_line;
   IF confirm = retrn THEN
      EXIT;
   END IF;
   Put ("ENTRY CANCELLED. REENTER FN:FT."); New_line;
   New_line;
END LOOP;

Delete_file (fcb1'ADDRESS, code);
Create_file (fcb1'ADDRESS, code);
fcb1.cr := byte (0);

--* SET DMA TO ADDRESS OF  DATA STRUCTURE THAT WILL  *--
--* HOLD DATA RECEIVED FROM SOURCE                   *--

Set_DMA (block'ADDRESS);
Clearscrn;
IF code = 255 THEN
   Put ("DIRECTORY SPACE UNAVAILABLE.");
ELSE
   Put ("FILE ");
   Outconsole (drive);
   Put (":");
   FOR i IN 1..fn_length LOOP
      Outconsole (fcb1.fn (i) );
   END LOOP;
   Put (".");
   FOR i IN 1..3 LOOP
```

```
      Outconsole (fcb1.ft (i) );
   END LOOP;
   Put (" IS OPENED."); New_line;

   --* RECEIVE DRIVE OF INCOMING DIRECTORY *--

   Drivein (drive);
   Put ("RECEIVING DIRECTORY FROM DRIVE ");
   Outconsole (drive); Put (" OF SENDING MACHINE...");
   New_line; New_line;
   LOOP

      --* RECEIVE IN 128 BYTE BLOCKS (8 ENTRIES)      *--
      --* UNTIL END OF DIRECTORY                       *--

      Erdfile (finished);
      EXIT WHEN finished;
      Recv_block (block'ADDRESS, length);
      IF block (1) /= byte (0) THEN

         --* NO MATCHES, DISPLAY NO DIRECTORY STRING *--

         FOR i IN 1..26 LOOP
            Outconsole (block (i));
         END LOOP;
         New_line;
      ELSE

         --* FORMAT EACH 8 DIRECTORY ENTRIES FOR      *--
         --* DISPLAY                                  *--

         start := 1; fini := 16;
         FOR i IN 1..8 LOOP
            FOR j IN start..fini LOOP
               EXIT WHEN block (start) = no;
               IF j = (start + 9) THEN
                  Outconsole (period);
               END IF;
               IF j > (start + 11) THEN
                  block (j) := space;
                  Outconsole (block (j));
               ELSE
                  Outconsole (block (j));
               END IF;
            END LOOP;
            IF i = 4 THEN
               New_line;
            END IF;
            start := start + 16; fini := fini + 16;
         END LOOP;
         New_line;
      END IF;
```

```
                New_line; New_line;
                Put ("PRESS RETURN IF YOU WANT TO SAVE ON FILE.");
                Keyin (choice);
                New_line; New_line;

                --* WRITE TO DISK ONLY USER'S CHOICES *--

                IF choice = retrn THEN
                   Write_seq (fcb1'ADDRESS, code);
                END IF;
             END LOOP;
             New_line; New_line;

             IF code = 1  THEN
                Put ("ERROR. NO AVAILABLE DIRECTORY SPACE.");
                New_line;
             ELSIF code = 2 THEN
                Put ("ERROR. DISK FULL."); New_line;
             ELSE
                Put ("FINISHED WRITING FILE."); New_line; New_line;
             END IF;

             Close_file (fcb1'ADDRESS, code);

          END IF;
          New_line;
          Put ("PRESS ANY KEY TO CONTINUE. ");
          Keyin (pause);

   END Receive_dir;

END Directry;

================================================================

PACKAGE Who IS
   PROCEDURE Whos_there;
END Who;



WITH Myasmlib, Myutil, Names;
PACKAGE BODY Who IS
   USE Myasmlib, Myutil, Names;

   PROCEDURE Whos_there IS

   --* AUTHOR: THOMAS V. WORKS
   --* DATE: SEPTEMBER 1986
   --* DESCRIPTION:  WHOS_THERE RECEIVES NET STATUS FROM THE  *--
   --* CONCENTRATOR.                                          *--
```

```
      machno: CONSTANT := 24;
      whos_on: ARRAY (1..machno) OF byte;
      number_on: Integer;
      ctrl_w: CONSTANT BYTE := byte (16#17#);

      BEGIN
         Clearscrn;
         Put ("FOR NET STATUS, ENTER YOUR MACHINE # FOR ");
         Put ("DESTINATION."); New_line; New_line;

         --* PROMPT USER FOR SOURCE TERMINAL # *--

         Enter_machine (dest, srce);

         --* CREATE CONNECTION RECORD *--

         connection.source := byte (srce);
         connection.destination := byte (dest);
         connection.process := ctrl_w;

         --* ESTABLISH CONNECTION WITH DESTINATION *--

         Setup (connection'ADDRESS, send_ready);

         --* RECEIVE NET STATUS AS A SINGLE BLOCK *--

         Recv_block (whos_on'ADDRESS, number_on);

         --* DISPLAY CURRENTLY ACTIVE TERMINAL #'s *--

         Put ("THE FOLLOWING NUMBERED MACHINES ARE CURRENTLY ");
         Put ("ACTIVE:"); New_line;
         FOR i IN 1..number_on LOOP
            Put_int (Integer (whos_on (i))); New_line;
         END LOOP;

   END Whos_there;

END Who;

================================================================

PACKAGE Bullbrd IS
   PROCEDURE Recv_bulletin;
END Bullbrd;
```

```
WITH Myasmlib, Xferfile, Myutil, Names;
PACKAGE BODY Bullbrd IS
   USE Myasmlib, Xferfile, Myutil, Names;

   PROCEDURE Recv_bulletin IS

   --* AUTHOR: THOMAS V. WORKS
   --* DATE: SEPTEMBER 1986
   --* DESCRIPTION:  RECV_BULLETIN PROMPT USER FOR FILE NAME  *--
   --* AND TYPE IN WHICH HE WISHES TO STORE MESSAGES RECEIVED *--
   --* FROM THE BULLETIN BOARD, OPENS THE FILE, RECEIVES DATA *--
   --* IN 128 BYTE BLOCKS, AND WRITES TO DISK THE RECEIVED    *--
   --* DATA.                                                  *--

      recv_b: CONSTANT BYTE := byte (16#02#);

      BEGIN
         Clearscrn;
         Put ("TO RECEIVE BULLETIN BOARD, ENTER 24 FOR ");
         Put ("DESTINATION MACHINE."); New_line;

         Enter_machine (dest, srce);

         --* CREATE CONNECTION RECORD *--

         connection.source := byte (srce);
         connection.destination := byte (dest);
         connection.process := recv_b;

         --* ESTABLISH CONNECTION WITH BULLETIN BOARD *--

         Put ("WAITING..."); New_line; New_line;
         Setup (connection'ADDRESS, send_ready);
         IF send_ready THEN
            Put ("CONNECTION ESTABLISHED. READY TO RECEIVE ");
            Put ("BULLETIN BOARD."); New_line; New_line;

            --* RECEIVE MESSAGES FROM BULLETIN BOARD AND      *--
            --* STORE ON FILE                                 *--

            Receivefile;
            Put ("BULLETIN BOARD RECEIVED."); New_line;
         ELSE
            Put ("BULLETIN BOARD INACTIVE."); New_line;
         END IF;
         Put ("PRESS ANY KEY TO CONTINUE.");
         Keyin (pause);

   END Recv_bulletin;

END Bullbrd;
```

```
==========================================================================

PACKAGE Myutil IS
   PROCEDURE Put_int (output: IN Integer);
   PROCEDURE Clearscrn;
   PROCEDURE Enter_machine (machdest: OUT Integer;
                            machsrce: OUT Integer);
   PROCEDURE Drive_select (d_drive: OUT byte);
END Myutil;



WITH Myasmlib, Names;
   PACKAGE BODY Myutil IS
   USE Myasmlib, Names;


   PROCEDURE Put_int (output: IN Integer) IS

   --* AUTHOR: THOMAS V. WORKS
   --* DATE: JUNE 1986
   --* DESCRIPTION:  PUT_INT DISPLAYS INTEGER VALUES BY        *--
   --* SEPARATING THE MOST SIGNIFICANT DIGIT AND DISPLAYING    *--
   --* IT UNTIL THERE ARE NO MORE DIGITS.                      *--

      max: Integer := 10000;
      count: Integer := 0;
      zero_ctr: Integer := 1;
      temp1, temp2: Integer;

      BEGIN
         temp1 := output;
         IF temp1 < 0 THEN
            Put ("-"); --* NEGATIVE NUMBER *--
         END IF;
         IF temp1 = 0 THEN
            Put ("0");
         ELSE
            WHILE max /= 0 LOOP
               LOOP

                  --* REMOVE MOST SIGNIFICANT DIGIT *--

                  temp2 := temp1/max;
                  temp1 := temp1 REM max;
                  count := count + 1;

                  --* REMOVE LEADING ZEROS *--

                  IF (count = zero_ctr) AND (temp2 = 0) THEN
                     zero_ctr := zero_ctr + 1;
                     max := max/10;
```

```
                  EXIT;
               END IF;

               --* DISPLAY MOST SIGNIFICANT DIGIT *--

               My_put (temp2);
               max := max/10;
               IF max = 0 THEN
                  EXIT;
               END IF;
            END LOOP;

         --* UNTIL THERE ARE NO MORE DIGITS *--

         END LOOP;
      END IF;

END Put_int;

-------------------------------------------------------------

PROCEDURE Clearscrn IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  CLEARSCRN DISPLAYS THE CLEAR SCREEN      *--
--* CODES CAUSING THE SCREEN TO BE CLEARED.                *--

   escape: CONSTANT BYTE := byte(16#1B#);
   clrscn: CONSTANT BYTE := byte(16#45#);

   BEGIN
      Outconsole (escape);
      Outconsole (clrscn);
      New_line; New_line;

END Clearscrn;

-------------------------------------------------------------

PROCEDURE Enter_machine (machdest: OUT Integer;
                         machsrce: OUT Integer) IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  ENTER_MACHINE PROMPTS USER FOR SOURCE    *--
--* AND DESTINATION TERMINAL NUMBERS AS A TWO DIGIT        *--
--* NUMBER AND THEN CONVERTS IT TO THE APPROPRIATE BYTE    *--
--* EQUIVALENT.                                            *--

   machine: ARRAY (1..3) OF byte;
   temp1, temp2: Integer;
```

```
BEGIN
   Put ("ENTER DESTINATION MACHINE (01,02..24) OR ");
   Put ("BROADCAST (00) "); New_line;
   Put ("FOLLOWED BY RETURN."); New_line;
   LOOP
      Put ("NOTE: BE SURE TO ADD LEADING ZERO."); New_line

      --* CONVERT TWO DIGIT KEYBOARD INPUT INTO BYTE     *-
      --* EQUIVALENT FOR DESTINATION TERMINAL            *-

      FOR i IN 1..3 LOOP
         Keyin (input);
         IF input = retrn THEN
            machine (i) := byte (16#30#);
            EXIT;
         ELSE
            machine (i) := input;
         END IF;
      END LOOP;

      Put ("MACHINE NUMBER ");
      FOR i IN 1..3 LOOP
         Outconsole (machine (i));
      END LOOP;
      Put (" IS SELECTED. PRESS RETURN TO CONFIRM.");
      New_line;
      Keyin (confirm);
      New_line;
      IF confirm = retrn THEN
         EXIT;
      END IF;
      Put ("ENTRY CANCELLED."); New_line;
      New_line;
   END LOOP;
   temp1 := Integer (machine (1)) - 16#30#;
   temp2 := Integer (machine (2)) - 16#30#;
   machdest := temp1 * 10 + temp2;

   LOOP
      Put ("ENTER YOUR MACHINE. NOTE: BE SURE TO ADD ");
      Put ("LEADING ZERO."); New_line;

      --* DITTO FOR SOURCE TERMINAL *--

      FOR i IN 1..3 LOOP
         Keyin (input);
         IF input = retrn THEN
            machine (i) := byte (16#30#);
            EXIT;
         ELSE
            machine (i) := input;
         END IF;
```

```
         END LOOP;

         Put ("MACHINE NUMBER ");
         FOR i IN 1..3 LOOP
            Outconsole (machine (i));
         END LOOP;
         Put (" IS SELECTED. PRESS RETURN TO CONFIRM.");
         New_line;
         Keyin (confirm);
         New_line;
         IF confirm = retrn THEN
            EXIT;
         END IF;
         Put ("ENTRY CANCELLED."); New_line;
         New_line;
      END LOOP;
      temp1 := Integer (machine (1)) - 16#30#;
      temp2 := Integer (machine (2)) - 16#30#;
      machsrce := temp1 * 10 + temp2;

END Enter_machine;

-------------------------------------------------------------

PROCEDURE Drive_select (d_drive: OUT byte) IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  DRIVE_SELECT PROMPTS USER FOR SELECTED   *--
--* DISK DRIVE AND PASSES INPUT TO OPERATING SYSTEM.       *--

   disk_drive: Integer;

   A: CONSTANT BYTE := byte (16#41#);
   B: CONSTANT BYTE := byte (16#42#);
   C: CONSTANT BYTE := byte (16#43#);
   D: CONSTANT BYTE := byte (16#44#);
   E: CONSTANT BYTE := byte (16#45#);

   --* LOWER CASE *--

   sa: CONSTANT BYTE := byte (16#61#);
   sb: CONSTANT BYTE := byte (16#62#);
   sc: CONSTANT BYTE := byte (16#63#);
   sd: CONSTANT BYTE := byte (16#64#);
   se: CONSTANT BYTE := byte (16#65#);

   BEGIN
      LOOP
         Put ("SELECT DRIVE: A, B, C, D, E.");
         Keyin (d_drive);
         New_line;
```

```
            Put ("DRIVE ");
            Outconsole (d_drive);
            Put (" IS SELECTED. PRESS RETURN TO CONFIRM, ");
            New_line;
            Put ("ANY OTHER KEY TO RESELECT.");
            Keyin (confirm);
            New_line;
            IF confirm = retrn THEN
               EXIT;
            END IF;
            Put ("ENTRY CANCELLED."); New_line;
            New_line;
         END LOOP;

         CASE d_drive IS
            WHEN A ! sa  =>
                        disk_drive := 0; Select_disk (disk_drive
            WHEN B ! sb  =>
                        disk_drive := 1; Select_disk (disk_drive
            WHEN C ! sc  =>
                        disk_drive := 2; Select_disk (disk_drive
            WHEN D ! sd  =>
                        disk_drive := 3; Select_disk (disk_drive
            WHEN E ! se  =>
                        disk_drive := 4; Select_disk (disk_drive
            WHEN OTHERS =>
                        Put ("INVALID DRIVE. DEFAULT IS A:");
                        New_line;
                        disk_drive := 0; Select_disk (disk_drive
         END CASE;
         New_line;

   END Drive_select;

END Myutil;

================================================================

PACKAGE Myasmlib IS
   PROCEDURE Create_file (address: IN Integer;
                          result: OUT Integer);
   PROCEDURE Close_file (address: IN Integer;
                         result: OUT Integer);
   PROCEDURE Open_file (address: IN Integer; result: OUT Integer);
   PROCEDURE Read_seq (address: IN Integer; result: OUT Integer);
   PROCEDURE Write_seq (address: IN Integer; result: OUT Integer);
   PROCEDURE Set_DMA (dma: IN Integer);
   PROCEDURE Delete_file (address: IN Integer;
                          result: OUT Integer);
   PROCEDURE Select_disk (disk: IN Integer);
   PROCEDURE Reset_disk;
   PROCEDURE Keyin (inchar : OUT byte);
```

```
   PROCEDURE Outconsole (outchar : IN byte);
   PROCEDURE Send_block (address: IN Integer; size: IN Integer);
   PROCEDURE Yes;
   PROCEDURE No;
   PROCEDURE Recv_block (address: IN Integer; len: OUT Integer);
   PROCEDURE Endfile (finish: OUT Boolean);
   PROCEDURE Active;
   PROCEDURE Waiting (stat: OUT byte);
   PROCEDURE Setup (addr: IN Integer; rdy: OUT Boolean);
   PROCEDURE My_put (inval: IN Integer);
   PROCEDURE Put_str (str: IN STRING);
   PROCEDURE Endmsg;
   PROCEDURE Search_first (address: IN Integer; buff: IN Integer;
                           result: OUT Integer; addr: OUT Integer);
   PROCEDURE Search_next (address: IN Integer; buff: IN Integer;
                          result: OUT Integer; addr: OUT Integer);
   PROCEDURE End_block;
   PROCEDURE Send_string (str: IN STRING);
   PROCEDURE Send_dir (dirad: IN Integer; size: IN Integer);
   PROCEDURE Driveout (driv: IN byte);
   PROCEDURE Drivein (driv: OUT byte);
   PROCEDURE Off;
END Myasmlib;



PACKAGE ASSEMBLY Myasmlib

   jmp main

   PROC Create_file;

   --* AUTHOR: THOMAS V. WORKS
   --* DATE: JUNE 1986
   --* DESCRIPTION:  CREATE_FILE CREATES A FILE SPECIFIED BY  *--
   --* THE FCB USING CP/M-86 FUNCTION #22.                    *--

   cr_code      equ 16h

            pop ax                  ;return address
            pop si                  ;code address
            pop dx                  ;FCB address
            push dx                 ;restore stack
            push si
            push ax
            mov cl, cr_code
            int 224
            mov [si], al
            ret

   END PROC Create_file;
```

```
PROC Close_file;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JUNE 1986
--* DESCRIPTION:  CLOSE_FILE CLOSES A FILE SPECIFIED BY     *-
--* THE FCB USING CP/M-86 FUNCTION #16.                     *-

cf_code      equ 10h

         pop ax                  ;return address
         pop si                  ;code address
         pop dx                  ;FCB address
         push dx                 ;restore stack
         push si
         push ax
         mov cl, cf_code
         int 224
         mov [si], al
         ret

END PROC Close_file;
-------------------------------------------------------------
PROC Open_file;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JUNE 1986
--* DESCRIPTION:  OPEN FILE OPENS A FILE SPECIFIED BY THE   *-
--* FCB USING CP/M-86 FUNCTION #15.                         *-

of_code      equ 0fh

         pop ax                  ;return address
         pop si                  ;code address
         pop dx                  ;FCB address
         push dx                 ;restore stack
         push si
         push ax
         mov cl, of_code
         int 224
         mov [si], al
         ret

END PROC Open_file;
-------------------------------------------------------------
PROC Read_seq;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JUNE 1986
--* DESCRIPTION:  READ_SEQ READS SEQUENTIAL 128 BYTE        *-
--* RECORDS FROM THE FILE SPECIFIED BY THE FCB USING        *-
--* CP/M-86 FUNCTION #20.                                   *-
```

```
rs_code      equ 14h

             pop ax                  ;return address
             pop si                  ;code address
             pop dx                  ;FCB address
             push dx                 ;restore stack
             push si
             push ax
             mov cl, rs_code
             int 224
             mov [si], ax
             ret

END PROC Read_seq;
-----------------------------------------------------------
PROC Write_seq;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JUNE 1986
--* DESCRIPTION:  WRITE_SEQ WRITES SEQUENTIAL 128 BYTE     *--
--* RECORDS TO THE FILE SPECIFIED BY THE FCB USING CP/M-86 *--
--* FUNCTION #21.                                          *--

ws_code      equ 15h

             pop ax                  ;return address
             pop si                  ;code address
             pop dx                  ;FCB address
             push dx                 ;restore stack
             push si
             push ax
             mov cl, ws_code
             int 224
             mov [si], ax
             ret

END PROC Write_seq;
-----------------------------------------------------------
PROC Set_DMA;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JUNE 1986
--* DESCRIPTION:  SET_DMA SETS THE DMA TO THE SPECIFIED    *--
--* ADDRESS USING CP/M-86 FUNCTION #26.                    *--

sdm_code     equ 1ah

             pop ax                  ;return address
             pop dx                  ;DMA address
             push ax                 ;restore stack
```

```
          mov cl, sdm_code
          int 224
          ret

END PROC Set_DMA;
-------------------------------------------------------------
PROC Delete_file;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JUNE 1986
--* DESCRIPTION:  DELETE_FILE DELETES THE FILE SPECIFIED   *-
--* BY THE FCB USING CP/M-86 FUNCTION #19.                 *-

df_code       equ 13h

          pop ax                  ;return address
          pop si                  ;code address
          pop dx                  ;FCB address
          push dx                 ;restore stack
          push si
          push ax
          mov cl, df_code
          int 224
          mov [si], al
          ret

END PROC Delete_file;
-------------------------------------------------------------
PROC Select_disk;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JULY 1986
--* DESCRIPTION:  SELECT_DISK DESIGNATES THE DEFAULT DISK  *-
--* DRIVE FOR SUBSEQUENT FILE OPERATIONS USING CP/M-86     *-
--* FUNCTION #14.                                          *-

sd_code      equ 0eh

          pop ax                  ;return address
          pop dx                  ;disk drive value
          push ax                 ;restore stack
          mov cl, sd_code
          int 224
          ret

END PROC Select_disk;
-------------------------------------------------------------
PROC Reset_disk;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JULY 1986
--* DESCRIPTION:  RESET_DISK RESETS ALL DISK DRIVES TO     *-
```

```
--* READ/WRITE AND SET THE A DISK AS THE DEFAULT DISK FOR  *--
--* ALL SUBSEQUENT FILE OPERATIONS USING CP/M-86 FUNCTION  *--
--* #13.                                                   *--

rd_code     equ 0dh

          mov cl, rd_code
          int 224
          ret

END PROC Reset_disk;
-------------------------------------------------------------
PROC Keyin;

--* AUTHOR: THOMAS V. WORKS
--* DATE: APRIL 1986
--* DESCRIPTION:  KEYIN OBTAINS INPUT FROM THE KEYBOARD    *--
--* USING CP/M-86 FUNCTION #06.                            *--

status_code     equ 0ffh
dirio_code      equ 06h
conout_code     equ 02h

          pop ax                ;return address
          pop di                ;output address
          push di               ;restore stack
          push ax
nokey:    mov cl, dirio_code
          mov dl, status_code
          int 224
          cmp al, 0             ;if zero, no input from keyboard
          jz nokey
          mov [di], al          ;store value from keyboard
          ret

END PROC Keyin;
PROC Outconsole;

--* AUTHOR: THOMAS V. WORKS
--* DATE: APRIL 1986
--* DESCRIPTION:  OUTCONSOLE DISPLAYS OUTPUT TO THE        *--
--* CONSOLE DEVICE USING CP/M-86 FUNCTION #02.             *--

          pop bx                ;return address
          pop ax                ;value to be output to console
          push bx               ;restore stack
          mov cl, conout_code
          mov dl, al
          int 224
          ret

END PROC Outconsole;
```

```
PROC Send_block;

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  SEND_BLOCK TRANSMITS A BLOCK OF DATA,     *-
--* SEQUENTIALLY ONE BYTE AT A TIME VIA THE MODEM PORT.     *-
--* EACH BYTE SENT IS ERROR CHECKED BY IMMEDIATE ECHO. THE *-
--* ADDRESS AND THE LENGTH OF THE DATA BLOCK TO BE          *-
--* TRANSMITTED ARE INPUT PARAMETERS.  A SEQUENCE OF FOUR  *-
--* BLOCK CODES (OFFh) INDICATING END OF BLOCK ARE SENT     *-
--* AFTER THE DATA.                                         *-

ctr             equ 03h
io_port         equ 0ECh
recv_rdy        equ 02h
error_code      equ 0EFh
block_code      equ 0FFh
status_port     equ 0EDh

          pop ax                ;return address
          pop bx                ;length of data structure
          pop si                ;data structure address
          push ax               ;restore stack

          mov ch, 4             ;block_code counter
          mov dl, io_port
loopb:    mov al, [si]          ;send out char
          out dx, al
          inc dx                ;statport
loopa:    in al, dx             ;wait for echo
          and al, recv_rdy
          jz loopa
          dec dx                ;dataport
          in al, dx             ;get echo
          cmp al, [si]          ;check for error
          jnz error1
          inc si                ;get new char
          dec bx                ;decrement length
          jnz loopb             ;do again until length = 0
          jmp over

error1:   mov al, error_code    ;tell receiver error in
                                ;transmission
          out dx, al
          inc dx                ;statport
loopd:    in al, dx             ;wait for echo
          and al, recv_rdy
          jz loopd
          dec dx                ;dataport
          in al, dx             ;check to see if error code wa
                                ;received

          cmp al, error_code
```

```
          jnz error1              ;if not, retransmit error code
error2:   mov al, error_code      ;send second error code
          out dx, al
          inc dx                  ;statport
loopd2:   in al, dx               ;wait for echo
          and al, recv_rdy
          jz loopd2
          dec dx                  ;dataport
          in al, dx               ;check to see if error code was
                                  ;received
          cmp al, error_code
          jnz error2              ;if not, retransmit error code
          jmp loopb               ;retransmit char

over:     mov al, block_code      ;tell receiver end of block
          out dx, al
          inc dx                  ;statport
loopc:    in al, dx               ;wait for echo
          and al, recv_rdy
          jz loopc
          dec dx                  ;dataport
          in al, dx               ;get echo
          cmp al, block_code      ;check for error
          jnz error3
          dec ch                  ;send out next block code
          jnz over                ;until four are sent out
          ret                     ;done

error3:   mov al, block_code
          out dx, al              ;retransmit
          inc dx                  ;statport
          jmp loopc               ;check again

END PROC Send_block;
-------------------------------------------------------------
PROC Yes;

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  YES TRANSMITS A SEQUENCE OF FOUR YES      *--
--* CODES (ØF1h) INDICATING END OF PROCESS.                 *--

yes_code        equ ØF1h

          pop ax
          push ax

          mov ch, 4               ;yes code counter
          mov dl, io_port         ;send out yes code
more2:    mov al, yes_code
          out dx, ax
          inc dx                  ;statport
```

```
loop2:   in al, dx              ;wait for echo
         and al, recv_rdy
         jz loop2
         dec dx                 ;dataport
         in al, dx              ;get echo
         cmp al, yes_code       ;check for error
         jnz error4             ;if error, retransmit
         dec ch                 ;do again until counter = 0
         jnz more2
         ret

error4:  mov al, yes_code       ;retransmit
         out dx, al
         inc dx                 ;statport
         jmp loop2

END PROC Yes;
-------------------------------------------------------------
PROC No;

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  NO TRANSMITS A SINGLE NO CODE (6EH)      *--
--* INDICATING PROCESS CONTINUING.                         *--

no_code        equ 6Eh

         pop ax
         push ax

         mov dl, io_port        ;send out no code
more3:   mov al, no_code
         out dx, ax
         inc dx                 ;statport
loop3:   in al, dx              ;wait for echo
         and al, recv_rdy
         jz loop3
         dec dx                 ;dataport
         in al, dx              ;get echo
         cmp al, no_code        ;check for error
         jnz more3              ;if error, do again
         ret

END PROC No;
PROC Recv_block;

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  RECV_BLOCK RECEIVES A BLOCK OF DATA,     *--
--* SEQUENTIALLY ONE BYTE AT A TIME VIA THE MODEM PORT.    *--
--* THE ADDRESS TO STORE THE RECEIVED THE DATA IS AN INPUT *--
--* PARAMETER AND THE AMOUNT OF DATA RECEIVED IS AN OUTPUT *--
```

```
--* PARAMETER.  RECV_BLOCK WAITS UNTIL IT RECEIVES A       *--
--* SEQUENCE OF FOUR BLOCK CODES (ØFFh) INDICATING END OF  *--
--* DATA BLOCK.                                            *--

        pop ax                  ;return address
        pop si                  ;length address
        pop di                  ;data structure address
        push di
        push si                 ;restore stack
        push ax

        mov bl, ctr             ;timeout counter
        mov ch, 4               ;finish_code counter
        mov cl, Ø               ;length counter
        mov dl, status_port
loop4:  in al, dx               ;check for incoming char
        and al, recv_rdy
        jz loop4
        dec dx                  ;dataport
        in al, dx               ;get char
        out dx, al              ;echo char
        mov [di], al            ;store char
        cmp al, error_code      ;check for error code
        jz error5
        inc di                  ;if no error, increment
                                ;location
        inc cl                  ;increment length
        cmp al, block_code      ;check for end of block
        jnz notyet
        dec ch                  ;begin counting block codes
        jz fini                 ;jump when number of block
                                ;codes = 4
        inc dx                  ;statport
        jmp loop4               ;if not check for next
                                ;finish code
notyet: mov ch, 4               ;reload finish code counter
        inc dx                  ;statport
        jmp loop4               ;get next char

error5: inc dx                  ;statport
loop5:  in al, dx               ;check for second error code
        and al, recv_rdy
        jz loop5
        dec dx                  ;dataport
        in al, dx               ;get input char
        out dx, al              ;echo input char
        cmp al, error_code      ;is this a true error
        jz eloop                ;if yes, jump to error handler
        mov [di], error_code    ;if not, treat it like a real
                                ;char and store
        inc di                  ;next location
        mov [di], al            ;store the char mistaken as
```

```
                                    ;second error code
            inc di                  ;next location - now we're
                                    ;back to normal
            inc dx                  ;statport
            jmp loop4               ;get next char

eloop:      inc dx                  ;statport
loop6:      in al, dx               ;check for retransmitted char
            and al, recv_rdy
            jz loop6
            dec dx                  ;dataport
            in al, dx               ;get retransmitted char
            out dx, al              ;echo retransmitted char
            dec di                  ;overwrite error
            mov [di], al
            inc di                  ;next location
            inc dx                  ;statport
            jmp loop4               ;get next char

fini:       inc dx                  ;statport
count1:     dec bl                  ;timeout to ensure echo was
                                    ;received
            jz none                 ;finish if counter = 0 and no
                                    ;char received
            in al, dx               ;wait for input
            and al, recv_rdy
            jz count1
            dec dx                  ;dataport
            in al, dx               ;get input
            out dx, al              ;echo input
            cmp al, block_code      ;one last check
            jz none                 ;if finish code ,continue
            mov bl, ctr             ;if not, do again
            jmp fini
none:       sub cl, 4               ;remove four block codes from
                                    ;length
            mov ch, 0               ;zero ch
            mov [si], cx            ;store length for return
            ret                     ;finished

END PROC Recv_block;
-------------------------------------------------------------------
PROC Endfile;

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  ENDFILE WAITS TO RECEIVE EITHER A SINGLE *--
--* NO CODE (0EEh) INDICATING FILE NOT FINISHED, OR A      *--
--* SEQUENCE OF FOUR YES CODES (0F1h) INDICATING FILE      *--
--* FINISHED.                                              *--
```

```
file_end        equ ØF1h
no_end          equ 6Fh

                pop ax                  ;return address
                pop di                  ;address of output variable
                push di                 ;restore stack
                push ax
                mov ch, 4               ;file end counter
                mov bl, ctr             ;timeout counter
more5:          mov dl, status_port     ;check for input
loop8:          in al, dx
                and al, recv_rdy
                jz loop8
                dec dx                  ;dataport
                in al, dx               ;get input
                out dx, al              ;echo input
                cmp al, no_end          ;decode input
                jz done1
                cmp al, file_end
                jnz more5               ;if neither, go back for
                                        ;retransmitted input
                dec ch                  ;if file end, begin counting
                                        ;until four are
                jnz more5               ;received
done1:          mov cl, al              ;store input temporarily
more6:          inc dx                  ;statport
count2:         dec bl                  ;timeout to ensure echo was
                                        ;received
                jz done2                ;finish if counter = Ø and no
                                        ;char received
                in al, dx               ;wait for input
                and al, recv_rdy
                jz count2
                dec dx                  ;dataport
                in al, dx               ;get input
                out dx, al              ;echo input
                cmp al, file_end        ;one last decode and check
                jz tru
                cmp al, no_end
                jz fal
                mov bl, ctr             ;if neither, do again
                jmp more6
done2:          mov al, cl              ;reload for decode
                cmp al, file_end        ;decode input
                jz tru
                cmp al, no_end
                jz fal
fal:            mov [di], ØØ            ;false
                ret
tru:            mov [di], Ø1            ;true
                ret
```

```
END PROC Endfile;

PROC Active;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  ACTIVE TRANSMITS THE ACTIVE CODE (ØD0h)  *--
--* TO THE CONCENTRATOR INDICATING AN ACTIVE STATUS AND    *--
--* WAITS FOR A REPLY.                                     *--

ready_code      equ ØD0h

        pop ax                  ;return address
        push ax                 ;restore stack

rdy:    mov dl, io_port         ;dataport
        mov al, ready_code
        out dx, al              ;send out ready code
        inc dx                  ;statport
loop9:  in al, dx               ;wait for echo
        and al, recv_rdy
        jz loop9
        dec dx                  ;dataport
        in al, dx               ;get echo
        cmp al, ready_code      ;check for error
        jnz rdy                 ;if not ready code, start
                                ;again
        ret                     ;if ready code, finish

END PROC Active;
-------------------------------------------------------------
PROC Waiting;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  WAITING AWAITS THE RESULT OF CHECK_QUEUE *--
--* THERE ARE FOUR POSSIBLE REPLYS: NOTHING (Ø), FILE      *--
--* WAITING (6), MESSAGE WAITING (ØDh), OR RESEND (1). THE *--
--* REPLY IS AN OUTPUT PARAMETER                           *--

        pop ax                  ;return address
        pop di                  ;status address
        push di                 ;restore stack
        push ax

        mov dl, status_port
loop1Ø: in al, dx               ;wait for input
        and al, recv_rdy
        jz loop1Ø
        dec dx                  ;dataport
        in al, dx               ;get input
        out dx, al              ;echo input
```

```
          mov [di], al          ;store input
more7:    inc dx                ;statport, wait for confirmation
          mov bl, ctr           ;load timer
count3:   dec bl
          jz finit              ;if no input and timer = 0,
                                ;then first input ok
          in al, dx             ;wait for confirmation
          and al, recv_rdy
          jz count3
          dec dx                ;oops there was an error
          in al, dx             ;get retransmitted input
          out dx, al            ;echo retransmitted input
          mov [di], al          ;store retransmitted input
          jmp more7             ;go back for confirmation
finit:    ret                   ;done

END PROC Waiting;
-------------------------------------------------------------
PROC Setup;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  SETUP TRANSMITS TO THE CONCENTRATOR THE  *--
--* CONNECTION RECORD, SEQUENTIALLY ONE BYTE AT A TIME.    *--
--* THEN IT WAITS FOR A REPLY; XFER (1), OR NO_XFER (0).   *--
--* THE REPLY IS A BOOLEAN OUTPUT PARAMETER                *--

          pop ax                ;return address
          pop di                ;sendrdy address
          pop si                ;data structure address
          push di               ;restore stack
          push ax

          mov bl, ctr           ;timeout counter
          mov cl, 2             ;length counter
          mov dl, io_port       ;dataport
start:    mov al, [si]          ;send destination out
          out dx, al
          inc dx                ;statport
here1:    in al, dx             ;wait for echo
          and al, recv_rdy
          jz here1
          dec dx                ;dataport
          in al, dx             ;get echo
          cmp al, [si]          ;check for error
          jnz fault
          inc si                ;go to next location
          dec cl                ;count length
          jnz start             ;jump until length = 2
          jmp finis             ;done
```

```
fault:  mov al, error_code      ;send out error code
        out dx, al
        inc dx                  ;statport
here2:  in al, dx               ;wait for echo
        and al, recv_rdy
        jz here2
        dec dx                  ;dataport
        in al, dx               ;get echo
        cmp al, error_code      ;check to see if error code wa
                                ;received
        jnz fault               ;if not, retransmit
        jmp start               ;retransmit data

finis:  mov al, block_code      ;send out block code
        out dx, al
        inc dx                  ;statport
here3:  in al, dx               ;wait for echo
        and al, recv_rdy
        jz here3
        dec dx                  ;dataport
        in al, dx               ;get echo
        cmp al, block_code      ;check to see if block code was
                                ;received
        jnz finis               ;if not, retransmit

        inc dx                  ;statport
lupe:   in al, dx               ;wait for sendrdy result
        and al, recv_rdy
        jz lupe
        dec dx                  ;dataport
        in al, dx               ;get result
        out dx, al              ;echo result
        mov [di], al            ;store result
        cmp al, 0               ;check for invalid result
        jz set
        cmp al, 1
        jz set
        mov [di], 0             ;if invalid, set sendrdy to
                                ;false
set:    inc dx                  ;statport
kount:  dec bl                  ;timeout to ensure echo receive
        jz allset
        in al, dx
        and al, recv_rdy
        jz kount
        dec dx                  ;dataport
        in al, dx               ;get input
        out dx, al              ;echo input
        mov [di], al            ;store input
        cmp al, 0               ;check for invalid result
        jz set
        cmp al, 1
```

```
        jz set
        mov [di], 0             ;if invalid, set sendrdy to
                                ;false
        mov bl, ctr             ;timeout again
        jmp kount

allset: ret

END PROC Setup;
-----------------------------------------------------------------
PROC My_put;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JULY 1986
--* DESCRIPTION:  MYPUT CONVERTS THE INTEGER INPUT          *--
--* PARAMETER TO ITS ASCII EQUIVALENT AND DISPLAYS IT.      *--

offset    equ 30h

        pop bx                  ;return address
        pop dx                  ;input value
        push bx                 ;restore stack

        mov cl, conout_code
        add dl, offset          ;convert to ascii for output
        int 224
        ret

END PROC My_put;
-----------------------------------------------------------------
PROC Put_str;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JULY 1986
--* DESCRIPTION:  PUT_STR DISPLAYS THE STRING INPUT. THE    *--
--* LENGTH OF THE STRING IS THE FIRST BYTE OF THE STRING.   *--

        pop ax                  ;return address
        pop si                  ;string address
        push ax                 ;restore stack

        mov al, [si]            ;first value is string length
        mov di, 0000            ;set di to zero
        mov ah, 00              ;set ah to zero
        mov di, ax              ;move string length to di
more8:  mov cl, conout_code
        inc si                  ;get next char
        mov dl, [si]
        int 224                 ;output to console
        dec di                  ;decrement string length
```

```
          cmp di, 0000
          jnz more8                ;finish when string length is
                                   ;zero
          ret

END PROC Put_str;
-------------------------------------------------------------------
PROC Endmsg;

--* AUTHOR: THOMAS V. WORKS
--* DATE: JULY 1986
--* DESCRIPTION:  ENDMSG TRANSMITS A SEQUENCE OF FOUR END  *--
--* OF MESSAGE CODES (0F1h) INDICATING END OF MESSAGE.     *--

msg_end          equ 0F1h

          pop ax                   ;return address
          push ax                  ;restore stack

          mov ch, 4                ;load message end counter
          mov bl, ctr              ;load timer
more9:    mov dl, status_port
loop11:   in al, dx                ;wait for message end code
          and al, recv_rdy
          jz loop11
          dec dx                   ;dataport
          in al, dx                ;get message end code
          out dx, al               ;echo input
          cmp al, msg_end          ;check for error
          jnz more9                ;if error, get next input
          dec ch                   ;count message end codes
          jnz more9                ;until counter = 0
more10:   inc dx                   ;statport
count4:   dec bl                   ;timeout to ensure echo receive
          jz done3
          in al, dx
          and al, recv_rdy
          jz count4
          dec dx                   ;dataport
          in al, dx                ;get input
          out dx, al               ;echo input
          cmp al, msg_end          ;one last check
          jz done3
          mov bl, ctr              ;if not message end code, time
                                   ;out again
          jmp more10

done3:    ret

END PROC Endmsg;
```

```
PROC Search_first;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  SEARCH_FIRST SEARCHES FOR THE FIRST       *--
--* INSTANCE OF DIRECTORY MATCH, READS 128 BYTE RECORD      *--
--* CONTAINING MATCHED ENTRY INTO DMA, AND COMPUTES OFFSET *--
--* TO ENTRY WITHIN 128 BYTE RECORD USING CP/M-86 FUNCTION *--
--* #17.                                                    *--

sf_code      equ 11h
finish_dir   equ 0ffh

          pop ax                ;return address
          pop si                ;dir_addr address
          pop di                ;code address
          pop bx                ;buffer address
          pop dx                ;fcb1 address
          push dx               ;restore stack
          push bx
          push di
          push si
          push ax

          mov [si], bx          ;save buffer address
          mov cl, sf_code
          int 224               ;search for first directory
                                ;match
          mov bx, [si]          ;restore buffer address
          mov [di], ax          ;store results of search in
                                ;code
          cmp al, finish_dir    ;are we at the end of the
                                ;directory
          jz done4              ;if yes, done
          mov cl, 32            ;directory match offset =
          mul cl                ;buffer address + (32 * search
                                ;result)
          add bx, ax
          mov [si], bx          ;store directory match offset
                                ;in dir_addr
done4:    ret

END PROC Search_first;
-------------------------------------------------------------
PROC Search_next;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  SEARCH_NEXT SEARCHES FOR THE NEXT         *--
--* INSTANCE OF DIRECTORY MATCH, READS 128 BYTE RECORD      *--
--* CONTAINING MATCHED ENTRY INTO DMA, AND COMPUTES OFFSET *--
```

```
--* TO ENTRY WITHIN 128 BYTE RECORD USING CP/M-86 FUNCTION *-
--* #18, UNTIL END OF DIRECTORY IS REACHED (0FFh).         *-

sn_code      equ 12h

             pop ax                  ;return address
             pop si                  ;dir_addr address
             pop di                  ;code address
             pop bx                  ;buffer address
             pop dx                  ;fcb1 address
             push dx                 ;restore stack
             push bx
             push di
             push si
             push ax

             mov [si], bx            ;save buffer address
             mov cl, sn_code
             int 224                 ;search for first directory
                                     ;match
             mov bx, [si]            ;restore buffer address
             mov [di], ax            ;store results of search
                                     ;in code
             cmp al, finish_dir      ;are we at the end of the
                                     ;directory
             jz done5                ;if yes, done
             mov cl, 32              ;directory match offset =
             mul cl                  ;buffer address + (32 * search
                                     ;result)
             add bx, ax
             mov [si], bx            ;store directory match offset
                                     ;in dir_addr
done5:       ret

END PROC Search_next;
-------------------------------------------------------------
PROC End_block;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  END_BLOCK TRANSMITS A SEQUENCE OF FOUR   *--
--* END OF BLOCK CODES (0FFh) INDICATING END OF BLOCK.     *--

             pop ax;                 ;return address
             push ax                 ;restore stack

             mov ch, 4               ;yes code counter
             mov dl, io_port         ;send out yes code
more12:      mov al, block_code
             out dx, ax
             inc dx                  ;statport
loop12:      in al, dx               ;wait for echo
```

```
          and al, recv_rdy
          jz loop12
          dec dx                    ;dataport
          in al, dx                 ;get echo
          cmp al, block_code        ;check for error
          jnz error6                ;if error, retransmit
          dec ch                    ;do again until counter = 0
          jnz more12
          ret

error6:   mov al, block_code        ;retransmit
          out dx, al
          inc dx                    ;statport
          jmp loop12

END PROC End_block;
-------------------------------------------------------------
PROC Send_string;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  SEND_STRING TRANSMITS A STRING INPUT,     *--
--* SEQUENTIALLY ONE BYTE AT A TIME.  THE ADDRESS OF THE    *--
--* STRING IS AN INPUT PARAMETERS.  THE LENGTH IS THE       *--
--* FIRST BYTE OF THE STRING.                               *--

          pop ax                    ;return address
          pop si                    ;string address
          push ax                   ;restore stack

          mov al, [si]              ;first char is string length
          mov  di, 0000             ;set up and store string length
                                    ;in di
          mov ah, 00
          mov di, ax
          mov dl, io_port
          inc si                    ;increment to first char

loop1b:   mov al, [si]              ;send out char
          out dx, al
          inc dx                    ;statport
loop1a:   in al, dx                 ;wait for echo
          and al, recv_rdy
          jz loop1a
          dec dx                    ;dataport
          in al, dx                 ;get echo
          cmp al, [si]              ;check for error
          jnz error7
          inc si                    ;get new char
          dec di                    ;decrement length
          jnz loop1b                ;do again until length = 0
          jmp done6
```

```
error7:  mov al, error_code     ;tell receiver error in
                                ;transmission
         out dx, al
         inc dx                 ;statport
loop1d:  in al, dx              ;wait for echo
         and al, recv_rdy
         jz loop1d
         dec dx                 ;dataport
         in al, dx              ;check to see if error code wa
                                ;received
         cmp al, error_code
         jnz error7             ;if not, retransmit error code
error8:  mov al, error_code     ;send second error code
         out dx, al
         inc dx                 ;statport
loop1c:  in al, dx              ;wait for echo
         and al, recv_rdy
         jz loop1c
         dec dx                 ;dataport
         in al, dx              ;check to see if error code wa
                                ;received
         cmp al, error_code
         jnz error8             ;if not, retransmit error code
         jmp loop1b             ;retransmit char

done6:   ret                    ;done

END PROC Send_string;
PROC Send_dir;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  SEND_DIR TRANSMITS A DIRECTORY ENTRY OF  *--
--* PREDEFINED LENGTH (BY CONSTANT INPUT PARAMETER). THE   *--
--* ADDRESS OF THE ENTRY IS THE OTHER INPUT PARAMETER.     *--

         pop ax                 ;return address
         pop bx                 ;data structure length
         pop si                 ;data structure address
         push ax
         mov dl, io_port        ;dataport
loop2b:  mov al, [si]           ;send out char
         out dx, al
         inc dx                 ;statport
loop2a:  in al, dx              ;wait for echo
         and al, recv_rdy
         jz loop2a
         dec dx                 ;dataport
         in al, dx              ;get echo
         cmp al, [si]           ;check for error
         jnz error9
```

```
          inc si                ;get new char
          dec bx                ;decrement length
          jnz loop2b            ;do again until length = 0
          jmp done7

error9:   mov al, error_code    ;tell receiver error in
                                ;transmission
          out dx, al
          inc dx                ;statport
loop2d:   in al, dx             ;wait for echo
          and al, recv_rdy
          jz loop2d
          dec dx                ;dataport
          in al, dx             ;check to see if error code was
                                ;received
          cmp al, error_code
          jnz error9            ;if not, retransmit error code
error10:  mov al, error_code    ;send second error code
          out dx, al
          inc dx                ;statport
loop2c:   in al, dx             ;wait for echo
          and al, recv_rdy
          jz loop2c
          dec dx                ;dataport
          in al, dx             ;check to see if error code was
                                ;received
          cmp al, error_code
          jnz error10           ;if not, retransmit error code
          jmp loop2b            ;retransmit char
done7:    ret                   ;done

END PROC Send_dir;
PROC Driveout;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  DRIVEOUT TRANSMITS THE BYTE EQUIVALENT   *--
--* OF THE DRIVE OF THE TRANSMITTED DIRECTORY.             *--

          pop ax                ;return address
          pop bx                ;drive value
          push ax               ;restore stack

dout:     mov dl, io_port       ;dataport
          mov al, bl
          out dx, al            ;send out drive
          inc dx                ;statport
loop13:   in al, dx             ;wait for echo
          and al, recv_rdy
          jz loop13
          dec dx                ;dataport
          in al, dx             ;get echo
```

```
          cmp al, bl              ;check for error
          jnz dout                ;if error, start again
          ret                     ;if not, finish

END PROC Driveout;
-------------------------------------------------------------
PROC Drivein;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  DRIVEIN RECEIVES THE BYTE EQUIVALENT OF  *--
--* THE DRIVE OF THE TRANSMITTED DIRECTORY.                *--

          pop ax                  ;return address
          pop di                  ;address for drive
          push di                 ;restore stack
          push ax

          mov bl, ctr             ;timeout counter
          mov dl, status_port
loop15:   in al, dx               ;wait for incoming go code
          and al, recv_rdy
          jz loop15
          dec dx                  ;dataport
          in al, dx               ;get input
          out dx, al              ;echo input
          mov [di], al            ;store input
more15:   inc dx                  ;statport
count5:   dec bl                  ;timeout to ensure echo was
                                  ;received
          jz done8                ;finish if counter = 0 and
                                  ;no char received
          in al, dx               ;wait for input
          and al, recv_rdy
          jz count5
          dec dx                  ;dataport
          in al, dx               ;get input
          out dx, al              ;echo input
          cmp al, [di]            ;one last check
          jz done8                ;if not error ,finish
          mov bl, ctr             ;if error, do again
          jmp more12
done8:    ret

END PROC Drivein;
-------------------------------------------------------------
PROC Off;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  OFF TRANSMITS THE OFF CODE (0Fh)         *--
```

```
   --* INDICATING TO THE CONCENTRATOR A TERMINAL IS NO LONGER *--
   --* ACTIVE.                                                 *--

   off_code        equ ØFh

            pop ax                  ;return address
            push ax                 ;restore stack

            mov dl, io_port         ;send out off code
            mov al, off_code
            out dx, al
            ret

   END PROC Off;

   main:

END Myasmlib;

================================================================

WITH Xferfile, Messages, Myasmlib, Directry,
     Who, Myutil, Bullbrd, Names;

PACKAGE BODY Xfermain IS
   USE Xferfile, Messages, Myasmlib, Directry,
       Who, Myutil, Bullbrd, Names;

   --****************--
   --* MAIN PROGRAM *--
   --****************--

   status: byte;
   ctrl_f: CONSTANT BYTE := byte (16#Ø6#);
   ctrl_m: CONSTANT BYTE := byte (16#ØD#);
   ctrl_d: CONSTANT BYTE := byte (16#Ø4#);
   resend: CONSTANT BYTE := byte (16#Ø1#);
   ctrl_l: CONSTANT BYTE := byte (16#ØC#);
   ctrl_r: CONSTANT BYTE := byte (16#12#);
   ctrl_s: CONSTANT BYTE := byte (16#13#);
   ctrl_t: CONSTANT BYTE := byte (16#14#);
   ctrl_x: CONSTANT BYTE := byte (16#18#);
   ctrl_w: CONSTANT BYTE := byte (16#17#);
   ctrl_g: CONSTANT BYTE := byte (16#Ø7#);
   ctrl_p: CONSTANT BYTE := byte (16#10#);
   ctrl_b: CONSTANT BYTE := byte (16#Ø2#);


   BEGIN
      Clearscrn;
      Put ("Welcome to the NPS Computer Science Labratory ");
      Put ("Local Area Network!");
```

```
New_line; New_line;
Put ("Execution Begins. Version 1.3"); New_line;
LOOP
   Put ("WAITING TO BE POLLED..."); New_line;

   --* SEND OUT ACTIVE CODE (ØD9h) AND WAIT FOR REPLY *--

   Active;
   Put ("CHECKING FOR INCOMING FILE OR MESSAGE."); New_lin
   Waiting (status);
   New_line;
   CASE status IS
      WHEN byte (Ø)
      => Put ("NO FILE OR MESSAGE INCOMING."); New_line;

      WHEN ctrl_f
      => Put ("FILE READY TO RECEIVE. PLEASE RECEIVE IT ")
         Put ("BEFORE PROCEDING."); New_line;
         Put ("FAILURE TO DO SO WILL CAUSE UNPREDICTABLE "
         Put ("RESULTS"); New_line;

      WHEN ctrl_m
      => Put ("MESSAGE READY TO RECEIVE. PLEASE RECEIVE ");
         Put ("IT BEFORE PROCEDING."); New_line;
         Put ("FAILURE TO DO SO WILL CAUSE UNPREDICTABLE "
         Put ("RESULTS"); New_line;

      WHEN ctrl_d
      => Put ("DIRECTORY READY TO RECEIVE. PLEASE RECEIVE"
         Put (" IT BEFORE PROCEDING."); New_line;
         Put ("FAILURE TO DO SO WILL CAUSE UNPREDICTABLE ")
         Put ("RESULTS"); New_line;

      WHEN resend
      => Put ("DESTINATION FOR YOUR PREVIOUS SESSION IS ");
         Put ("NOW ACTIVE. PLEASE RESEND."); New_line;
         Put ("FAILURE TO DO SO WILL CAUSE UNPREDICTABLE ")
         Put ("RESULTS"); New_line;

   END CASE;
   New_line;

   --* USER MAIN MENU *--

   Put ("CTRL_S = SEND FILE "); New_line;
   Put ("CTRL_R = RECEIVE FILE "); New_line;
   Put ("CTRL_T = SEND MESSAGE (TALK) "); New_line;
   Put ("CTRL_L = RECEIVE MESSAGE (LISTEN) "); New_line;
   Put ("CTRL_P = SEND DIRECTORY "); New_line;
   Put ("CTRL_G = RECEIVE DIRECTORY "); New_line;
   Put ("CTRL_W = GET NET STATUS (WHO) "); New_line;
   Put ("CTRL_B = RECEIVE BULLETIN BOARD "); New_line;
```

```
            New_line;
            New_line;
            Put ("ENTER INPUT: ");
            Keyin (input);
            New_line;
            CASE input IS
               WHEN ctrl_s => Sendfile;
               WHEN ctrl_r => Receivefile;
               WHEN ctrl_t => Talking;
               WHEN ctrl_l => Listening;
               WHEN ctrl_w => Whos_there;
               WHEN ctrl_g => Receive_dir;
               WHEN ctrl_p => Present_dir;
               WHEN ctrl_b => Recv_bulletin;
               WHEN OTHERS => NULL;
            END CASE;
            Put ("CTRL_X = EXIT. ANYTHING ELSE TO CONTINUE.");
            Keyin (input);
            EXIT WHEN input = ctrl_x;
            Clearscrn;

         --* LOOP CONTINUOUSLY UNTIL USER EXITS *--

         END LOOP;

         --* TELL CONCENTRATOR NO LONGER ACTIVE *--

         Off;

END Xfermain;

================================================================

WITH Myasmlib, Myutil, Names;
PACKAGE BODY Bulletin IS
   USE Myasmlib, Myutil, Names;

   --** BULLETIN BOARD **--

   max_msg_length: CONSTANT := 1600;
   max_msg: CONSTANT := 20;
   receive: CONSTANT BYTE := byte (16#0D#);  --CTRL_M--
   send: CONSTANT BYTE := byte (16#02#);  --CTRL_B--

   TYPE msg IS
      RECORD
         message: ARRAY (1..max_msg_length) OF byte;
         msg_length: Integer;
      END RECORD;

   msg_list: ARRAY (1..max_msg) OF msg;
```

```
command: byte;
msg_count: Integer := 0;
byte_count: Integer := 0;
msg_block: ARRAY (1..block_size) OF byte;


BEGIN
   Clearscrn;
   Put ("NPS COMPUTER SCIENCE LABRATORY BULLETIN BOARD.");
   New_line; New_line;
   LOOP

      --* TELL CONCENTRATOR BULLETIN BOARD ACTIVE AND WAIT *--
      --* FOR REPLY                                       *--

      Active;

      --* MESSAGES INCOMING OR REQUEST TO SEND? *--

      Waiting (command);
      IF command = receive THEN
         msg_count := msg_count + 1;

         --* RECEIVE MESSAGES UNTIL MSG_COUNT = 20 *--

         Recv_block (msg_list(msg_count).message'ADDRESS,
                     msg_list(msg_count).msg_length);
         Endmsg;
         Put ("*");
      ELSIF command = send THEN

         --* SEND ALL MESSAGES CURRENTLY ON BOARD *--

         FOR i IN 1..msg_count LOOP
            byte_count := 0;
            LOOP

               --* FORMAT MESSAGES IN 128 BYTE BLOCKS FOR  *--
               --* TRANSMISSION. FOR EACH MESSAGE SEND     *--
               --* UNTIL BYTE_COUNT = MSG_LENGTH THEN ADD  *--
               --* BLANKS AS NECESSARY TO FILL IN LAST 128 *--
               --* BYTE BLOCK. SEND NEXT MESSAGE UNTIL ALL *--
               --* MESSAGES ARE SENT                       *--

               FOR j IN 1..block_size LOOP
                  byte_count := byte_count + 1;
                  IF byte_count > msg_list(i).msg_length THEN
                     msg_block (j) := space;
                  ELSE
                     msg_block (j) :=
                              msg_list(i).message(byte_count);
                  END IF;
```

```
                    END LOOP;
                    No;
                    Send_block (msg_block'ADDRESS, block_size);
                    EXIT WHEN byte_count > msg_list(i).msg_length;
                END LOOP;

            END LOOP;  --MSG_COUNT--

            Yes;
        END IF;  --COMMAND--

    END LOOP;  --MAIN--

END Bulletin;
```

==========

The following batch file (xfer.sub) is used to compile the preceding programs:

```
era myutil.sym
era myasmlib.sym
era xferfile.sym
era messages.sym
era directry.sym
era who.sym
era bullbrd.sym
era names.sym
era myutil.jrl
era myasmlib.jrl
era xferfile.jrl
era messages.jrl
era directry.jrl
era who.jrl
era bullbrd.jrl
era xfermain.jrl
era xfermain.cmd
janus names.spc
janus myutil.spc
janus myasmlib.spc
janus xferfile.spc
janus messages.spc
janus directry.spc
janus who.spc
janus bullbrd.spc
janus myutil
jasm86 myasmlib
janus xferfile
janus messages
janus directry
```

```
janus who
janus bullbrd
janus xfermain
jlink xfermain
era bulletin.jrl
era bulletin.cmd
janus bulletin
jlink bulletin
```

APPENDIX E

LISTING OF CONCENTRATOR PROGRAMS

```
PACKAGE Concname IS

   --* GLOBAL TYPES, CONSTANTS, AND VARIABLES *--

   TYPE process_status IS
      RECORD
         sourceport: Integer;
         destport: Integer;
         process_type: byte;
      END RECORD;

   max_que: CONSTANT := 552;  --FOR 24 TERMINALS--

   TYPE que IS ARRAY (1..max_que) OF process_status;

   queue: que;

   resend: CONSTANT BYTE := byte (16#01#);
   zero: CONSTANT BYTE := byte (0);
   machno: CONSTANT := 24;
   ready: Boolean;

   active_list: ARRAY (1..machno) OF Boolean;

END Concname;

================================================================

PACKAGE Concutil IS
   PROCEDURE Check_queue (number_que: IN Integer;
                          port: IN Integer;
                          que_id: OUT Integer;
                          result_que: OUT Boolean);
   PROCEDURE Net_stat (destination: IN Integer);
END Concutil;



WITH Coasmlib, Concname;
PACKAGE BODY Concutil IS
   USE Coasmlib, Concname;

   PROCEDURE Check_queue (number_que: IN Integer;
                          port: IN Integer;
                          que_id: OUT Integer;
                          result_que: OUT Boolean) IS
```

```
--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  CHECK_QUEUE CHECKS THE QUEUE FOR WAITING *-
--* PROCESSES AND TELLS SOURCE AND DESTINATION WHAT, IF    *-
--* ANYTHING, IS WAITING.  BOOLEAN RESULT OF THE CHECK IS  *-
--* AN OUTPUT PARAMETER.                                   *-

   BEGIN
      IF number_que = 0 THEN

         --* NOTHING IN QUEUE *--

         result_que := false;
         Queue_status (port, zero);
      ELSE
         FOR i IN 1..(number_que + 1) LOOP
            IF queue (i).destport = port THEN
               que_id := i;
               result_que := true;
               EXIT;
            END IF;
            que_id := i;
         END LOOP;
         IF que_id = (number_que + 1) THEN

            --* NOTHING IN QUEUE FOR POLLED PORT *--

            result_que := false;
            Queue_status (port, zero);
         ELSE
            FOR i IN 1..255 LOOP
               FOR j IN 1..255 LOOP

                  --* DELAY FOR SOURCE *--

                  NULL;
               END LOOP;
            END LOOP;

            --* POLL SOURCE OF WAITING PROCESS *--

            Check_port (queue(queue_id).sourceport, ready);

            IF ready THEN

               --* TELL POLLED PORT WHAT IS WAITING FOR IT *--

               Queue_status (port, queue(que_id).process_type)

               --* TELL SOURCE OF WAITING PROCESS TO RESEND *-
```

```
                Queue_status (queue(que_id).sourceport, resend);
             ELSE

                --* TELL POLLED PORT TO GO AHEAD *--

                result_que := false;
                Queue_status (port, zero);
             END IF;
          END IF;
       END IF;

END Check_queue;

-----------------------------------------------------------------

PROCEDURE Net_stat (destination: IN Integer) IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  NET_STAT CHECKS THE ACTIVE_LIST FOR      *--
--* ACTIVE PORTS, PLACES LIST OF ACTIVE PORTS IN WHO_LIST, *--
--* AND TRANSMITS DATA TO REQUESTING TERMINAL.             *--

   wnum: Integer := 0;
   who_length: Integer := 0;

   who_list: ARRAY (1..machno) OF byte;

   BEGIN
      FOR i IN 1..machno LOOP
         IF active_list (i) = true THEN

            --* PLACE ACTIVE PORTS IN WHO LIST *--

            wnum := wnum + 1;
            who_list (wnum) := byte (i);
            who_length := who_length + 1;
         END IF;
      END LOOP;

      Send_who_block (destination, who_list'ADDRESS,
                      who_length);

END Net_stat;

END Concutil;

=================================================================

PACKAGE Coasmlib IS
   PROCEDURE Check_port (prt: IN Integer; rdy: OUT Boolean);
   PROCEDURE Connect (prt: IN Integer; addr: IN Integer);
```

```
   PROCEDURE Queue_status (prt: IN Integer; proc: IN byte);
   PROCEDURE Send_who_block (dest: IN Integer; addr: IN Integer;
                             len: IN Integer);
   PROCEDURE Broadcast (s_prt: IN Integer; t_input: OUT byte);
   PROCEDURE Concxfer (s_prt: IN Integer; d_prt: IN Integer);
   PROCEDURE Xfer (s_prt: IN Integer);
   PROCEDURE No_xfer (s_prt: IN Integer);
END Coasmlib;



PACKAGE ASSEMBLY Coasmlib

   jmp main

   PROC Check_port;

   --* AUTHOR: THOMAS V. WORKS
   --* DATE: SEPTEMBER 1986
   --* DESCRIPTION:  CHECK_PORT CHECKS THE PORTS OF THE       *--
   --* NETWORK LOOKING FOR THE ACTIVE CODE (ØDØh). IT TIMES   *--
   --* OUT IF THERE IS NO RESPONSE.  CHECK_PORT ALSO          *--
   --* DETERMINES IF A DESTINATION PORT IS ACTIVE. THE        *--
   --* BOOLEAN RESULT IS THE OUTPUT PARAMETER.                *--

   ready_code      equ ØD0h
   recv_rdy        equ Ø2h
   timer           equ 10h

            pop ax                  ;return address
            pop di                  ;data address
            pop dx                  ;dataport
            push dx
            push di                 ;restore stack
            push ax

   again:   inc dx                  ;statport
            mov bl, timer           ;load timer
   loop3:   dec bl
            jz nxtprt               ;go to next port if no input
            in al, dx               ;check for input
            and al, recv_rdy
            jz loop3
            dec dx                  ;dataport
            in al, dx               ;get input
            out dx, al              ;echo input
            cmp al, ready_code      ;check for error
            jnz again               ;if error, get new input
   more:    inc dx                  ;statport
            mov bl, timer           ;load timer
   time:    dec bl                  ;timer to ensure echo received
            jz thsprt               ;go to this port if timer = Ø
```

```
          in al, dx             ;check for input
          and al, recv_rdy
          jz time
          dec dx                ;dataport
          in al, dx             ;get input
          out dx, al            ;echo input
          cmp al, ready_code    ;check for error
          jnz more              ;if error get new input
thsprt:   mov [di], Ø1          ;set ready to true
          ret
nxtprt:   mov [di], ØØ          ;set ready to false
          ret

END PROC Check_port;
-------------------------------------------------------------
PROC Connect;

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  CONNECT RECEIVES THE CONNECTION RECORD   *--
--* FROM SETUP.                                            *--

error_code      equ ØEFh
finish_code     equ ØFFh

          pop ax                ;return address
          pop di                ;data structure address
          pop dx                ;dataport
          push ax               ;restore stack

          inc dx                ;statport
here4:    in al, dx             ;wait for data
          and al, recv_rdy
          jz here4
          dec dx                ;dataport
          in al, dx             ;get data
          out dx, al            ;echo data
          cmp al, error_code    ;check for error code
          jz errors
          cmp al, finish_code   ;check for finish code
          jz done
          mov [di], al          ;store data
          inc di                ;go to next location
          inc dx                ;statport
          jmp here4             ;get next data

errors:   dec di                ;go back to previous position
          inc dx                ;statport
here5:    in al, dx             ;wait for retransmitted data
          and al, recv_rdy
          jz here5
          dec dx                ;dataport
```

```
          in al, dx              ;get retransmitted data
          out dx, al             ;echo retransmitted data
          mov [di], al           ;overwrite bad data
          inc di                 ;go to next location
          inc dx                 ;statport
          jmp here4              ;get next data

done:     ret

END PROC Connect;

PROC Queue_status;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  QUEUE_STATUS TELLS THE SOURCE AND          *--
--* DESTINATION PORTS ABOUT THE PROCESSES WAITING IN THE     *--
--* QUEUE.                                                   *--

          pop ax                 ;return address
          pop dx                 ;dataport
          pop si                 ;value of queue status
          push ax                ;restore stack

redo:     mov ax, si
          out dx, ax             ;send out queue status
          inc dx                 ;statport
loop5:    in al, dx              ;wait for echo
          and al, recv_rdy
          jz loop5
          dec dx                 ;dataport
          in al, dx              ;get echo
          cmp ax, si             ;check for error
          jnz redo               ;if error, retransmit
          ret                    ;if not, finish

END PROC Queue_status;
-------------------------------------------------------------
PROC Send_who_block;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  SEND_WHO_BLOCK TRANSMITS THE LIST OF       *--
--* ACTIVE PORTS FOLLOWED BY A SEQUENCE OF FOUR FINISH       *--
--* CODES (0FFh) INDICATING END OF LIST TO THE REQUESTING    *--
--* TERMINAL.                                                *--

          pop ax                 ;return address
          pop bx                 ;length of data structure
          pop si                 ;data structure address
          pop dx                 ;destination port (data)
          push ax                ;restore stack
```

```
        mov ch, 4               ;finish_code counter
loopb:  mov al, [si]            ;send out char
        out dx, al
        inc dx                  ;statport
loopa:  in al, dx               ;wait for echo
        and al, recv_rdy
        jz loopa
        dec dx                  ;dataport
        in al, dx               ;get echo
        cmp al, [si]            ;check for error
        jnz error1
        inc si                  ;get new char
        dec bx                  ;decrement length
        jnz loopb               ;do again until length = 0
        jmp over

error1: mov al, error_code      ;error in transmission
        out dx, al
        inc dx                  ;statport
loopd:  in al, dx               ;wait for echo
        and al, recv_rdy
        jz loopd
        dec dx                  ;dataport
        in al, dx               ;was error code received
        cmp al, error_code
        jnz error1              ;if not, retransmit error code
error2: mov al, error_code      ;send second error code
        out dx, al
        inc dx                  ;statport
loopd2: in al, dx               ;wait for echo
        and al, recv_rdy
        jz loopd2
        dec dx                  ;dataport
        in al, dx               ;was error code  received
        cmp al, error_code
        jnz error2              ;if not, retransmit error code
        jmp loopb               ;retransmit char

over:   mov al, finish_code     ;end of transmission
        out dx, al
        inc dx                  ;statport
loopc:  in al, dx               ;wait for echo
        and al, recv_rdy
        jz loopc
        dec dx                  ;dataport
        in al, dx               ;get echo
        cmp al, finish_code     ;check for error
        jnz error3
        dec ch                  ;send out next finish code
        jnz over                ;until four are sent out
        ret                     ;done
```

```
error3:  mov al, finish_code
         out dx, al              ;retransmit
         inc dx                  ;statport
         jmp loopc               ;check again

END PROC Send_who_block;
-------------------------------------------------------------
PROC Broadcast;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  BROADCAST TRANSMITS DATA RECEIVED FROM   *--
--* THE SOURCE TO ALL ACTIVE DESTINATION TERMINALS,        *--
--* BYPASSING THE SOURCE AND THE BULLETIN BOARD PORTS.  A  *--
--* BYTE IS RECEIVED FROM THE SOURCE AND TRANSMITTED TO    *--
--* EACH DESTINATION, THEN BROADCAST CHECKS EACH ECHO FOR  *--
--* ERROR.  BROADCAST FINISHES WHEN IT RECEIVES A SEQUENCE *--
--* OF FOUR FINISH CODES (0F1h) FROM EACH DESTINATION.     *--

portnum          equ 08h
boardnum         equ 03h
ctr              equ 04h
portone          equ 0100h
bullport         equ 019Ch

         pop ax                  ;return address
         pop di                  ;temporary input holder
         pop si                  ;source port (data)
         push si                 ;restore stack
         push di
         push ax

         mov ch, 0               ;load error code counter
         mov cl, ctr             ;load finish code counter
         mov bl, portnum
         mov bh, boardnum
         mov dx, si              ;load source port
nextch:  inc dx                  ;statport
l_one:   in al, dx               ;wait for input
         and al, recv_rdy
         jz l_one
         dec dx                  ;dataport
         in al, dx               ;get input
         mov [di], al            ;store input
         mov dx, portone         ;load first destination port
l_two:   cmp dx, si              ;do not send to source port
         jz skip
         cmp dx, bullport        ;or to bulletin board port
         jz skip
         out dx, al              ;send input
skip:    add dx, 4               ;next destination port
```

```
          dec bl
          jnz l_two              ;continue sending until all
                                 ;ports are addressed
          add dx, 32             ;next board
          mov bl, portnum        ;reload
          dec bh
          jnz l_two              ;continue sending until all
                                 ;boards are addressed
          mov bl, portnum        ;reload for echo checking
          mov bh, boardnum
          mov dx, portone        ;load first destination port
l_three:  cmp dx, si             ;do not check source port
          jz next
          cmp dx, bullport       ;or bulletin board
          jz next
          inc dx                 ;statport
          in al, dx              ;wait for echo
          dec dx                 ;dataport
          and al, recv_rdy
          jz next                ;go to next port if no response
          in al, dx              ;get echo
          cmp al, [di]           ;check for error
          jnz errorb
next:     add dx, 4              ;next destination port
          dec bl
          jnz l_three            ;continue receiving until all
                                 ;ports are addressed
          add dx, 32             ;next board
          mov bl, portnum        ;reload
          dec bh
          jnz l_three            ;continue receiving until all
                                 ;boards are addressed
          mov bh, boardnum       ;reload
          cmp ch, 0
          jz cont                ;if no errors, continue

                                 ;ERROR HANDLER
loop10:   pop dx                 ;get next error address (data)
loop11:   inc dx                 ;statport
l_six:    in al, dx              ;wait for echo
          and al, recv_rdy
          jz l_six
          dec dx                 ;dataport
          in al, dx              ;get echo
          cmp al, error_code     ;check for error
          jnz reerror
          mov al, [di]           ;if no error, retransmit char
                                 ;input
          out dx, al
          inc dx                 ;statport
l_seven:  in al, dx              ;wait for echo
          and al, recv_rdy
```

```
          jz l_seven
          dec dx                    ;dataport
          in al, dx                 ;get echo
          cmp al, [di]              ;check for error in
                                    ;retransmission
          jnz reerror
          dec ch                    ;if we finally got it right, go
                                    ;to next error address
          jnz loop1Ø                ;until ch = Ø
          jmp cont                  ;continue

reerror:  mov al, error_code        ;resend error code
          out dx, al
          jmp loop11

errorb:   push dx                   ;save address where error
                                    ;occurred
          inc ch                    ;count addressess
          mov al, error_code
          out dx, al                ;send out error code
          jmp back                  ;return to check other ports

                                    ;CONTINUE
cont:     mov al, [di]              ;send echo back to source
          mov dx, si
          out dx, al
          cmp al, end_process       ;check for end of process
          jnz no_end
          dec cl                    ;begin counting end process
                                    ;codes
          jz finis                  ;until four in a row are
                                    ;received
          jmp nextch
no_end:   mov cl, ctr               ;if not, reload and
          jmp nextch                ;get next char input

finis:    ret

END PROC Broadcast;
-------------------------------------------------------------
PROC Concxfer;

--* AUTHOR: THOMAS V. WORKS
--* DATE: AUGUST 1986
--* DESCRIPTION:  CONCXFER TRANSMITS DATA RECEIVED FROM    *--
--* THE SOURCE TO THE ACTIVE DESTINATION TERMINAL. A BYTE  *--
--* IS RECEIVED FROM THE SOURCE AND TRANSMITTED TO THE     *--
--* DESTINATION, WITHOUT ERROR CHECKING. ERROR CHECKING IS *--
--* DONE BY THE TERMINALS. CONCXFER FINISHES WHEN IT       *--
--* RECEIVES A SEQUENCE OF FOUR FINISH CODES (ØF1h) FROM   *--
--* THE DESTINATION.                                       *--
```

```
end_process     equ ØF1h

          pop ax                ;return address
          pop cx                ;destport (data)
          pop bx                ;sourceport (data)
          push ax               ;restore stack

          mov di, 4             ;finish code counter
          mov dx, bx            ;sourceport (data)
start:    inc dx   0            ;statport
loop1:    in al, dx             ;check for incoming char
          and al, recv_rdy
          jz loop1
          dec dx                ;dataport
          in al, dx             ;get incoming char
          mov dx, cx            ;destport (data)
          out dx, al            ;send char to receiver
          inc dx                ;statport
loop2:    in al, dx             ;check for echo
          and al, recv_rdy
          jz loop2
          dec dx                ;dataport
          in al, dx             ;get echo
          mov dx, bx            ;sourceport (data)
          out dx, al            ;send echo to transmitter
          cmp al, end_process   ;if end process code, begin
                                ;counting
          jnz notyet            ;if not, get next char
          dec di                ;until four end process codes
                                ;are received
          jz fini
          jmp start
notyet:   mov di, 4             ;reload finish code counter
          jmp start

fini:     ret

END PROC Corcxfer;
-------------------------------------------------------------
PROC Xfer;

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  XFER TRANSMITS THE XFER CODE (1) TO THE  *--
--* SOURCE INDICATING THAT THE DESTINATION TERMINAL IS     *--
--* ACTIVE.                                                *--

          pop ax                ;return address
          pop dx                ;sourceport (data)
          push ax               ;restore stack
```

```
    xmit1:   mov al, 01              ;set send_ready to true
             out dx, al
             inc dx                  ;statport
    loopx1:  in al, dx               ;wait for echo
             and al, recv_rdy
             jz loopx1
             dec dx                  ;dataport
             in al, dx               ;get echo
             cmp al, 01              ;check for error
             jnz xmit1               ;if error, retransmit

             ret

    END PROC Xfer;
    ------------------------------------------------------------
    PROC No_xfer;

    --* AUTHOR: THOMAS V. WORKS
    --* DATE: SEPTEMBER 1986
    --* DESCRIPTION:  NO_XFER TRANSMITS THE NO_XFER CODE (0)   *--
    --* TO THE SOURCE INDICATING THAT THE DESTINATION TERMINAL *--
    --* IS NOT ACTIVE.                                         *--

             pop ax                  ;return address
             pop dx                  ;sourceport (data)
             push ax                 ;restore stack

    xmit2:   mov al, 00              ;set send_ready to false
             out dx, al
             inc dx                  ;statport
    loopx2:  in al, dx               ;wait for echo
             and al, recv_rdy
             jz loopx2
             dec dx                  ;dataport
             in al, dx               ;get echo
             cmp al, 00              ;check for error
             jnz xmit2               ;if error, retransmit

             ret

    END PROC No_xfer;

    main:

END Coasmlib;

============================================================

WITH Coasmlib, Concutil, Concname;
PACKAGE BODY Poll IS
   USE Coasmlib, Concutil, Concname;
```

```
--*****************************--
--* MAIN CONCENTRATOR PROGRAM *--
--*****************************--

TYPE conctn IS
   RECORD
      process: byte;
      source: byte;
      destination: byte;
   END RECORD;

connection: conctn;

ctrl_f: CONSTANT BYTE := byte (16#06#);
ctrl_m: CONSTANT BYTE := byte (16#0D#);
ctrl_d: CONSTANT BYTE := byte (16#04#);
ctrl_w: CONSTANT BYTE := byte (16#17#);
ctrl_b: CONSTANT BYTE := byte (16#02#);

boardno: CONSTANT := 3;
portno:  CONSTANT := 8;
firstport: CONSTANT := 16#100#;
lastport: CONSTANT := 16#19C#; --BULLETIN BOARD--

pollport: Integer;
inqueue: Boolean;
numqueue: Integer := 0;
index: Integer := 0;
new_queue: Integer;
machine_count: Integer := 0;
input: byte;

PROCEDURE Convert (idx: IN Integer) IS

--* AUTHOR: THOMAS V. WORKS
--* DATE: SEPTEMBER 1986
--* DESCRIPTION:  CONVERT CONVERTS THE CONNECTION RECORD   *--
--* RECEIVED FROM THE TERMINAL TO PHYSICAL PORT ADDRESSES. *--

   BEGIN

      CASE queue(idx).destport IS
         WHEN 1 => queue(idx).destport := 16#100#;
         WHEN 2 => queue(idx).destport := 16#104#;
         WHEN 3 => queue(idx).destport := 16#108#;
         WHEN 4 => queue(idx).destport := 16#10C#;
         WHEN 5 => queue(idx).destport := 16#110#;
         WHEN 6 => queue(idx).destport := 16#114#;
         WHEN 7 => queue(idx).destport := 16#118#;
         WHEN 8 => queue(idx).destport := 16#11C#;
         WHEN 9 => queue(idx).destport := 16#140#;
         WHEN 10 => queue(idx).destport := 16#144#;
```

```
            WHEN 11 => queue(idx).destport := 16#148#;
            WHEN 12 => queue(idx).destport := 16#14C#;
            WHEN 13 => queue(idx).destport := 16#150#;
            WHEN 14 => queue(idx).destport := 16#154#;
            WHEN 15 => queue(idx).destport := 16#158#;
            WHEN 16 => queue(idx).destport := 16#15C#;
            WHEN 17 => queue(idx).destport := 16#180#;
            WHEN 18 => queue(idx).destport := 16#184#;
            WHEN 19 => queue(idx).destport := 16#188#;
            WHEN 20 => queue(idx).destport := 16#18C#;
            WHEN 21 => queue(idx).destport := 16#190#;
            WHEN 22 => queue(idx).destport := 16#194#;
            WHEN 23 => queue(idx).destport := 16#198#;
            WHEN 24 => queue(idx).destport := 16#19C#;
            WHEN OTHERS => NULL;
         END CASE;

         CASE queue(idx).sourceport IS
            WHEN 1 => queue(idx).sourceport := 16#100#;
            WHEN 2 => queue(idx).sourceport := 16#104#;
            WHEN 3 => queue(idx).sourceport := 16#108#;
            WHEN 4 => queue(idx).sourceport := 16#10C#;
            WHEN 5 => queue(idx).sourceport := 16#110#;
            WHEN 6 => queue(idx).sourceport := 16#114#;
            WHEN 7 => queue(idx).sourceport := 16#118#;
            WHEN 8 => queue(idx).sourceport := 16#11C#;
            WHEN 9 => queue(idx).sourceport := 16#140#;
            WHEN 10 => queue(idx).sourceport := 16#144#;
            WHEN 11 => queue(idx).sourceport := 16#148#;
            WHEN 12 => queue(idx).sourceport := 16#14C#;
            WHEN 13 => queue(idx).sourceport := 16#150#;
            WHEN 14 => queue(idx).sourceport := 16#154#;
            WHEN 15 => queue(idx).sourceport := 16#158#;
            WHEN 16 => queue(idx).sourceport := 16#15C#;
            WHEN 17 => queue(idx).sourceport := 16#180#;
            WHEN 18 => queue(idx).sourceport := 16#184#;
            WHEN 19 => queue(idx).sourceport := 16#188#;
            WHEN 20 => queue(idx).sourceport := 16#18C#;
            WHEN 21 => queue(idx).sourceport := 16#190#;
            WHEN 22 => queue(idx).sourceport := 16#194#;
            WHEN 23 => queue(idx).sourceport := 16#198#;
            WHEN 24 => queue(idx).sourceport := 16#19C#;
            WHEN OTHERS => NULL;
         END CASE;

   END Convert;


   BEGIN
      LOOP  --MAIN--
      pollport := firstport;
      machine_count := 0;
```

```
Outer:
   FOR i IN 1..boardno LOOP
      FOR j IN 1..portno LOOP
         machine_count := machine_count + 1;
         active_list(machine_count) := false;
         Check_port (pollport, ready);
         IF ready THEN
            active_list(machine_count) := true;

            --* CHECK QUEUE FOR WAITING PROCESSES FOR   *--
            --* POLLED PORT                            *--

            Check_queue (numqueue, pollport, index,
                         inqueue);

            IF inqueue THEN

            --* SERVICE WAITING PROCESS FIRST           *--
            --* ESTABLISH CONNECTION WITH SOURCE OF     *--
            --* WAITING PROCESS                        *--

               Connect (queue(index).sourceport,
                        connection'ADDRESS);

               --* DECODE CONNECTION RECORD *--

               queue(index).sourceport :=
                              Integer (connection.source);
               queue(index).destport :=
                         Integer (connection.destination);
               queue(index).process_type :=
                                   connection.process;
               Convert (index);

               --* TELL SOURCE TO SEND *--

               Xfer (queue(index).sourceport);

               --* ESTABLISH DATA COMMUNICATION CHANNEL *--

               Concxfer (queue(index).sourceport,
                         queue(index).destport);

               --* REMOVE PROCESS FROM QUEUE *--

               numqueue := numqueue - 1;
               FOR m IN 1..255 LOOP
                  FOR n IN 1..255 LOOP
                     NULL;
               --* DELAY TO ALLOW RECEIVER TO GET READY *--
               --* TO SEND                             *--
                  END LOOP;
```

```
      END LOOP;
      Check_port (pollport, ready);

      --* IF POLLED PORT DOES NOT SEND IN  *--
      --* TIME, RESUME POLLING FROM START  *--
      EXIT Outer WHEN NOT ready;

      --* WHEN WAITING PROCESS IS FINISHED *--
      --* TELL POLLED PORT NOTHING WAITING *--
      --* AND PROCESS ORIGINAL REQUEST     *--
      Queue_status (pollport, zero);
   END IF; --INQUEUE--

   --* PROCESS NEW REQUEST *--
   --* ESTABLISH CONNECTION WITH SOURCE OF *--
   --* ORIGINAL REQUEST                    *--

   Connect (pollport, connection'ADDRESS);

   --* LATEST QUEUE POSITION  *--

   new_queue := numqueue + 1;

   --* DECODE CONNECTION *--

   queue(new_queue).sourceport :=
                        Integer (connection.source);
   queue(new_queue).destport :=
                   Integer (connection.destination);
   queue(new_queue).process_type :=
                              connection.process;
   Convert (new_queue);

   IF connection.destination = byte (0) THEN

      --* POLL ALL DESTINATION PORTS EXCEPT *--
      --* SOURCE AND BULLETIN BOARD         *--

      pollport := firstport;
      FOR k IN 1..portno LOOP
         IF (pollport /=
                        queue(new_queue).sourceport)
                     AND (pollport /= lastport) THEN
            Check_port (pollport, ready);
            IF ready THEN

               --* TELL DESTINATION *--

               queue_status (pollport,
                             connection.process);
            END IF;
         END IF;
```

```
            pollport := pollport + 4;
         END LOOP;

         FOR n IN 1..255 LOOP
            FOR m IN 1..255 LOOP
               NULL; --* DELAY TO ALLOW RECEIVER *--
                     --* TO GET READY           *--
            END LOOP;
         END LOOP;

         --* TELL SOURCE TO BROADCAST *--

         Xfer (queue(new_queue).sourceport);
         Broadcast (queue(new_queue).sourceport,
                    input);

         --* RESET POLLPORT AND RESUME POLLING *--

         pollport := queue(new_queue).sourceport;
      ELSIF connection.process = ctrl_w THEN

         --* TELL SOURCE TO RECEIVE NET STATUS *--
         --* THEN SEND NET STATUS             *--

         Xfer (pollport);
         Net_stat (pollport);
      ELSE

         --* PROCESS NORMAL REQUEST *--

         --* IS DESTINATION ACTIVE? *--

         Check_port (queue(new_queue).destport,
                     ready);

         IF ready THEN

            --* TELL DESTINATION WHAT IS COMING *--

            Queue_status (queue(new_queue).destport,
                          queue(new_queue).process_type);

            --* TELL SOURCE TO SEND *--

            Xfer (pollport);
            IF connection.process = ctrl_b THEN

               --* ESTABLISH CONNECTION WITH *--
               --* BULLETIN BOARD           *--

               Concxfer (queue(new_queue).destport,
                         queue(new_queue).sourceport);
```

```
                        ELSE

                           --* ESTABLISH DATA COMMUNICATION *--
                           --* CHANNEL                      *--

                           Concxfer (queue(new_queue).sourceport,
                                     queue(new_queue).destport);
                        END IF;
                     ELSE

                        --* TELL SOURCE, DESTINATION INACTIVE *--

                        No_xfer (pollport);
                        IF (queue(new_queue).process_type = ctrl_
                        OR (queue(new_queue).process_type = ctrl_
                        THEN

                           --* PLACE PROCESS IN QUEUE *--
                           --* IF FILE OR MESSAGE     *--

                           numqueue := numqueue + 1;
                        END IF;
                     END IF;
                  END IF;
               END IF; --READY--

               pollport := pollport + 4;
               EXIT WHEN pollport = lastport;

               --* DON'T POLL BULLETIN BOARD *--

            END LOOP; --INNER FOR LOOP--

            pollport := pollport + 32;
         END LOOP Outer;

      END LOOP; --MAIN LOOP--

END Poll;
```

==========

The following batch file (conc.sub) is used to compile the preceeding programs:

```
era coasmlib.sym
era concutil.sym
era concname.sym
era concutil.jrl
era coasmlib.jrl
era poll.jrl
era poll.cmd
```

```
janus concname.spc
janus coasmlib.spc
janus concutil.spc
jasm86 coasmlib
janus concutil
janus poll
jlink poll
```

APPENDIX F

GLOSSARY

1. Z-100

Z-120 is the specific model name for the microcomputers used in this network. The vendor is Zenith Data Systems.

2. SBC

SBC is an acronym for Single Board Computer. It is a configuration of VLSI (Very Large Scale Integration) circuitry on one computer board capable of performing the functions of a computer. The SBC is the driving force in the Concentrator. The vendor of the 86/12A SBC used in this thesis is Intel Corporation.

3. Concentrator

The Concentrator is the collection of hardware and software that performs the network switching functions.

4. MULTIBUS

MULTIBUS is the specific model name for the hardware that allows multiple SBC's to communicate directly with common memory.

5. Local Area Network

A local Area Network is any network that operates exclusively within a low radius (max 50 miles) region; usually a single building or a group of buildings.

6. Workstation

A Workstation is viewed as the Z-100 microcomputer and associated peripheral devices that perform the application functions of the network.

7. Process

Processes are viewed as the transfer of files, messages, or directorys between Z-100 workstations. Each process has a sending and a receiving function.

8. Communication

Communication is viewed as transmitting data and/or commands between the sending and receiving functions of a process.

9. Data Communication Channel

Data Communication Channel refers to the channel or 'pathway' used to transfer intra-process communications between Z-100 workstations.

10. Circuit Switching

Circuit Switching refers to the method of network communication whereby the entire message containing data and commands is transmitted from the sender to the receiver along a dedicated communication channel.

11. Ports

Ports are channels through which processes communicate.

12. USART

USART is an acronym for Universal Synchronous/ Asynchronous Receiver/Transmitter. It is a microprocessor that provides communication interface between computers or between a computer and a peripheral device.

13. NPS

NPS is an acronym for Naval Postgraduate School, Monterey, California.

LIST OF REFERENCES

1. Hartman, R.L. and Yasinsac, A.F., Janus/Ada Implementation of a Star Cluster Network of Personal Computers With Interface to an ETHERNET LAN Allowing Access to DDN Resources, M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1986.

2. Digital Research, P/M-86 Systems Guide, 1981.

3. Digital Research, CP/M-86 Programmer's Guide, 1981.

4. Klien, D., "Will Ada Succeed?" Defense Electronics, Volume 16, pp. 105-108, December 1984.

5. Bernard, E.V., "Ada Steps Out," Datamation, Volume 29, pp. 114-128, September 1983.

6. RR Software Inc., Janus/Ada Package User Manuals, 1983.

7. Barnes, J.G.P., Programing in Ada, Addison-Wesley, Menlo Park, 1984.

BIBLIOGRAPHY

Choi, K.J. and Lee, J.K., Printer Multiplexing Among Multiple Z-100 Microcomputers, M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1985.

Intel Corporation, iSBC 86/12A Single Board Computer Hardware Reference Manual, 1979.

National Semiconductor Corporation, BLC 8534/8538 4 and 8 Channel Communication Expansion Boards Hardware Reference Manual, 1980.

Zenith Data Systems Corporation, Z-100 User's Guide, 1982.

Zenith Data Systems Corporation, Z-100 Technical Manuals, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Department Chairman, Code 54 Department of Administrative Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Computer Technology Programs Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Dr. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	3
6.	Professor Michael P. Spencer, Code 54Sp Department of Administrative Science Naval Postgraduate School Monterey, California 93943-5000	1
7.	LT Thomas V. Works 1576 Fruitdale Drive San Jose, California 95124	3
8.	Daniel Green, Code 20F Naval Surface Weapons Center Dahlgren, Virginia 22449	1
9.	CAPT J. Donegan, USN Naval Sea Systems Command Washington, D.C. 20362	1

10. PCA AEGIS Data Repository 1
RCA Corporation
Government Systems Department
Mail Stop 127-327
Moorestown, New Jersey 08057

11. Library (Code F33-05) 1
Naval Surface Warfare Center
Dahlgren, Virginia 22449

12. Dr. M. J. Gralia 1
Applied Physics Laboratory
John Hopkins Road
Laurel, Maryland 20707

13. Dana Small 1
Code 8242, NOSC
San Diego, California 92152

Works
JANUS/Ada software implementation of a star cluster local area network of personal computers.

Thesis
W8751
c.1

Works
JANUS/Ada software implementation of a star cluster local area network of personal computers.